FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Jogos Sérios para Ensino de Engenharia de Software

Edgar Alves

Mestrado Integrado em Engenharia Informática e Computação

Orientador: João Carlos Pascoal de Faria (Doutor)

Junho de 2013

© Edgar Alves, 2013

Jogos Sérios para Ensino de Engenharia de Software

Edgar Alves

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Rosaldo Rossetti (Doutor)

Vogal Externo: Daniel Silva (Doutor)

Orientador: João Carlos Pascoal de Faria (Doutor)

---

29 de Julho de 2013

Resumo

Desde o aparecimento dos jogos de computador que o impacto deles na sociedade tem vindo a aumentar. Cada vez mais se veem crianças, adolescentes e até mesmo adultos que despendem tempo da sua vida com jogos de computador. Isto deve-se ao facto dos jogos terem a capacidade de captar a atenção do jogador, de o estimular e desafiar, chegando por vezes até a ser viciantes. Dada a importância dos jogos de computador na sociedade, surgiu a ideia de se criar jogos que, para além da componente de entretenimento, incluíssem também uma componente didática. O objetivo desta combinação é, sobretudo, aproveitar a capacidade dos jogos como focos de atenção e entretenimento, e acrescentar-lhes uma fonte de conhecimento tornando-os num método mais agradável de aprendizagem. Estes jogos, que combinam entretenimento e aprendizagem, chamam-se jogos sérios.

O presente trabalho de dissertação teve precisamente como objetivo, criar um jogo sério aplicado ao ensino de Engenharia de *Software* de forma a incentivar os alunos do ensino superior a estudarem através de métodos de estudo mais agradáveis. A aplicação desenvolvida, de nome SCOREL, ou Source COde REview Learning, consiste num jogo que permite agilizar e incentivar a prática de revisão de código, uma das muitas tarefas da Engenharia de *Software*. Permite também que os utilizadores façam o carregamento dos seus próprios exercícios de revisão de código para que, sejam posteriormente utilizados por outros utilizadores. Para a realização deste trabalho foi importante fazer um estudo inicial do estado da arte relativamente aos jogos sérios aplicados na Engenharia de *Software* para averiguar o que já existe na área.

Abstract

Since the advent of computer games their impact on society is increasing. Increasingly one sees children, teenagers and even adults who spend time playing computer games. This is due to the fact that games have the ability to capture the attention of the player, to stimulate and challenge him, even to be addictive sometimes. Given the importance of computer games in society, arises the idea of creating games that in addition to the entertainment component would include a learning component as well. The aim of this combination is primarily harnessing the ability of games as focal points and entertainment and add them a source of knowledge making them a more enjoyable method of learning. These games, that combine entertainment and learning, are called serious games.

This dissertation work aims to create a serious game applied to Software Engineering education in order to encourage higher education students to study through nicer methods. The developed application named SCOREL, or Source COde REview Learning, is a game that allows one to streamline and encourage the practice of code review, one of the many tasks of Software Engineering. It also allows that users upload their own exercises of code revision to be used by the other users. For this project it was important to make an initial study of the state of the art relative to serious games used in Software Engineering to find out what already exists in this area.

Agradecimentos

Agradecimento ao orientador desta tese, professor João Carlos Pascoal de Faria pela dedicação e atenção prestada, sempre acompanhando o desenvolvimento do trabalho de dissertação com sugestões e ideias visando o aperfeiçoamento do mesmo. Agradecimento também, ao aluno e colega de curso, Amaro Silva, pelo seu apoio e conselhos de nível técnico na fase inicial do projeto e ainda ao professor Miguel Pimenta Monteiro pelo esclarecimento de algumas dúvidas e partilha de conhecimento relativamente à arquitetura do mesmo.

Edgar Alves

Índice

Lista de Ilustrações

Lista de Tabelas

Abreviaturas e Símbolos

API	Application Programming Interface
DGBL	Digital Game-based Learning
FEUP	Faculdade de Engenharia da Universidade do Porto
FPS	First Person Shooter
GBL	Game-based Learning
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
JAX-WS	Java API for XML Web Services
JDBC	Java Database Connectivity
RPC	Remote Procedure Call
RPG	Role-Playing Game
SCOREL	Source COde REview Learning
SW	Software
SWEBOK	Guide to the Software Engineering Body of Knowledge
UDK	Unreal Development Kit
V&V	Verificação e Validação
WSDL	Web Services Description Language
XML	eXtensible Markup Language

1. Introdução

Este documento descreve o trabalho de dissertação desenvolvido e nele são apresentados os resultados da aplicação denominada SCOREL, ou Source COde REview Learning, uma aplicação desenvolvida com o propósito de inovar os métodos de estudo de Revisão de Código uma das tarefas de Verificação e Validação de *software*.

Neste capítulo é feito um enquadramento e uma pequena descrição do trabalho de dissertação desenvolvido e qual a motivação e objetivos do mesmo. Por fim é explicado como o documento se encontra estruturado.

1.1. Enquadramento

Este trabalho foi desenvolvido no âmbito da Dissertação do Mestrado Integrado em Engenharia Informática e Computação na Faculdade de Engenharia da Universidade do Porto que está entre as vinte melhores da Europa e as cem melhores de todo o mundo (Ranking Web of Universities 2013). É conhecida por ser uma faculdade com um grande grau de exigência e por isso uma faculdade com bastante prestígio tanto a nível nacional como a nível internacional.

O objetivo desta dissertação é o desenvolvimento de um jogo sério para o ensino de Engenharia de *Software* e surge com o propósito de inovar os métodos de estudo tradicionais. O estudo tradicional, acima de tudo quando associado a um ensino exigente como é o superior, tem, por vezes, uma conotação negativa como atividade aborrecida e maçadora. E, por isso, muitas vezes, os alunos estudam por obrigação e por necessidade sendo poucos os que gostam dessa tarefa. Por outro lado, jogar jogos de computador é uma atividade imediatamente associada a emoções como motivação e diversão. É uma atividade que geralmente qualquer pessoa gosta por ser desafiante e motivadora, sendo por isso muito cativante e um grande foco de atenção para quem a pratica. Portanto, a inclusão de jogos nos métodos de ensino, os

chamados jogos sérios, é uma tentativa de tornar o estudo numa tarefa mais agradável combinando a capacidade de entreter que os jogos têm com uma componente de ensino.

1.2. Motivação e Objetivos

Hoje em dia, cada vez mais se vê a utilização de jogos em situações de bastante seriedade. Exemplo disso são os simuladores, considerados jogos, que permitem encenar situações muito próximas da realidade. Este tipo de jogos têm a grande vantagem de permitirem experimentação sem risco possibilitando assim o treino e aperfeiçoamento das técnicas antes da utilização das mesmas na vida real onde a falha, por vezes, pode sair cara. O facto de já existirem exemplos de jogos sérios de grande sucesso noutras áreas é a grande motivação para este projeto. Apesar deste tipo de jogos já abranger um variado leque de áreas (Frye 2010), a disciplina de Engenharia de *Software* não está ainda muito explorada havendo áreas de conhecimento desta que, segundo a pesquisa efetuada, não estão ainda exploradas. Este trabalho de dissertação tem como objetivo, o desenvolvimento de uma solução que aborde uma destas áreas. Assim, a solução desenvolvida consiste numa aplicação Java que tem como principal funcionalidade a Revisão de Código, uma das tarefas da Verificação e Validação, sob a forma de um jogo.

Em síntese, a tese que se pretende demonstrar com o desenvolvimento desta aplicação, é que é possível automatizar exercícios para o treino de práticas de engenharia de *software,* como as práticas de revisão de código, e ao mesmo tempo introduzir componentes de um jogo sério, nomeadamente, a competição entre utilizadores.

A automatização dos exercícios consiste num sistema que permite uma avaliação e *feedback* automático dado pelo computador ao utilizador. Este sistema de avaliação revelou-se algo desafiante no seu desenvolvimento pois utiliza uma ferramenta de comparação de *strings* para a verificação das soluções. Isto tem os seus inconvenientes pois a marcação de erros através da seleção de texto pode, por vezes, ser subjetiva e por isso foi preciso criar um sistema de avaliação o mais robusto possível. Para evitar estes problemas de subjetividade, este documento apresenta no capítulo 6 um tutorial de marcação de erros apresentando o padrão que deve ser utilizado tanto pelos autores como pelos jogadores.

A inclusão de componentes de jogos sérios, nomeadamente, a competição entre utilizadores, como foi referido anteriormente, consiste na utilização de um sistema de pontuações e tabela de melhores resultados, que incentiva a realização de cada vez mais exercícios de forma a subir na tabela classificativa. O sistema de pontuações utilizado será explicado com maior detalhe na secção 4.4.

A metodologia de desenvolvimento escolhida para este trabalho, consistiu numa abordagem iterativa com uma definição incremental de requisitos e de opções de implementação segundo prioridades estabelecidas. Optou-se por este método por se tratar de um projeto de inovação.

1.3. Estrutura do Documento

Este documento encontra-se estruturado em dez capítulos. Após a Introdução, segue-se o capítulo intitulado Conceitos Básicos. Neste, é feita uma descrição de conceitos integrantes do mesmo, nomeadamente, jogos de computador e Engenharia de *Software*. De seguida sucede-se o capítulo da Revisão Bibliográfica onde é apresentada parte do trabalho de pesquisa elaborado relativamente ao Estado da Arte. Seguem-se depois, os capítulos Definição do Conceito do Jogo e Desenho e Implementação do projeto. Após o capítulo da Desenho e Implementação é explicado o modo de funcionamento da solução desenvolvida e resultados da experimentação no capítulo Utilização e Experimentação. Este documento termina então com o capítulo de Conclusões e Trabalho Futuro seguido das Referências e dos Anexo.

2. Conceitos Básicos

O presente capítulo aborda o problema deste projeto sendo apresentadas em maior detalhe as áreas integrantes do mesmo, nomeadamente os jogos de computador e a Engenharia de *Software*.

2.1. Jogos

Um jogo representa uma atividade lúdica que engloba um conjunto de várias componentes. Entre elas há três obrigatórias que são:

Jogador – um ou mais elementos que procuram diversão partilhando ou disputando objetivos caso sejam parceiros de equipa ou adversários, respetivamente;

Regras – definem o modo como o jogo deve ser jogado indicando as permissões e restrições do jogo;

Objetivos – conjunto de elementos ou estado do jogo que os jogadores tentam alcançar.

O jogo tem a capacidade de incutir no jogador a vontade de superar o desafio superando o(s) adversários(s) ou superando-se a si mesmo. É por isso uma atividade que além de divertida, é por vezes viciante sendo um grande foco de atenção para os jogadores (Wikipédia - Jogo 2013).

2.2. Jogos de Computador

Os jogos de computador são jogos digitais que têm como plataforma um computador. À semelhança dos jogos tradicionais, estes englobam o mesmo conjunto de componentes obrigatórios. Um jogo de computador pode estar inserido num vasto conjunto de tipos de jogos, de entre os quais se podem destacar por exemplo os *Role-Playing Games (RPGs)*, os *First*

Person Shooters (FPSs), jogos de estratégia, jogos de desporto, jogos de combate, jogos de condução entre outros.

2.3. Jogos Sérios

Nem todos os jogos têm apenas a finalidade de divertir e entreter o jogador. Na década de oitenta surgiu o primeiro jogo onde o propósito principal não era a diversão mas sim a transmissão de conteúdo de cariz educativo (Wikipédia - Serious game 2013). A estes jogos dá-se o nome de jogos sérios, do termo inglês *Serious Games*.

Eles são chamados jogos sérios, não porque os outros jogos não o são, mas sim porque estes são usados de uma forma pedagógica para fins políticos, sociais, publicitários, económicos, ambientais ou causas humanitárias (Arvers 2009).

A utilização de jogos para fins educacionais tem como principal objetivo transmitir informação de uma forma mais agradável e deste modo mais eficaz também. Na geografia - que é quase ignorada nos dias de hoje - não há razão para uma geração que consegue memorizar mais de 100 personagens Pokémon incluindo as suas características, história e evolução, não consiga aprender os nomes, populações, capitais e relações de todas as 101 nações do mundo. Depende apenas de como é apresentado (Prensky 2001a). Talvez a diferença mais importante é que o "material" a ser aprendido – informação, conceitos, relações, etc. – não pode simplesmente ser "dito" a estas pessoas. Tem de ser aprendido por elas próprias, através de questões, descoberta, construção, interação e, acima de tudo, diversão (Prensky 2001b).

Os jogos sérios servem então para combater a desmotivação dos alunos relativamente às práticas estudantis aborrecidas e dessa forma obterem melhores resultados.

Estes jogos estão frequentemente ligados a áreas como defesa, educação, exploração científica, serviços de saúde, engenharia entre outros. Segundo as áreas onde os jogos sérios se inserem, estes podem ser classificados de vários tipos. A classificação dos jogos sérios não está ainda muito sólida mas há termos comuns para alguns tipos existentes. De entre eles destacam-se os indicados na Tabela 1.

Tabela 1 Tipos de jogos sérios (Wikipédia - Serious game 2013; Serious Game University 2013)

Designação (inglês)	Descrição
Advergames	Jogo utilizado para promover uma marca, um produto, uma organização ou um ponto de vista.
Edutainment	Jogo de entretenimento que é projetado para ser educativo.
Game-based Learning	Jogo com objetivos de aprendizagem. São projetados de forma a equilibrar a componente lúdica com a componente didática.
Newsgames	Jogo jornalístico que reporta acontecimentos recentes ou

	envia comentários editoriais sobre o acontecimento.
Training and Simulation Games	Jogo que simula atividades da vida real com a maior exatidão possível. Utilizado para adquirir ou exercitar diferentes habilidades.
Persuasive Games	Jogo que influencia os jogadores a agirem através da jogabilidade. É projetado para mudar atitudes e comportamentos dos utilizadores através da persuasão.
Organizational-dynamic	Jogo projetado com propósitos específicos de promover o desenvolvimento pessoal e formação de caráter. Aborda questões de dinâmica organizacional, em três níveis: comportamento individual, dinâmica de grupo e rede e dinâmica cultural.
Games for Health	Jogo utilizado para ensaios médicos, educação de saúde, terapia psicológica e reabilitação física ou cognitiva.
Art Games	Jogo utilizado para expressar ideias artísticas ou arte produzida através de jogos digitais.
Militainment	Jogo utilizado para fins militares que replica operações com um alto grau de precisão.

2.4. *Game-Based Learning* e *Digital Game-Based Learning*

GBL é o tipo de jogo sério mais relacionado com a solução pretendida para o presente projeto uma vez que este tipo de jogos sérios tem como objetivo a aprendizagem. Neste caso, como se trata de um jogo de computador, o termo correto será DGBL que no fundo assenta na definição do GBL mas tendo a atenção que se refere a jogos digitais.

GBL utiliza exercícios competitivos, colocando os estudantes uns contra os outros ou a desafiarem-se a si próprios, a fim de motivá-los a aprender melhor (Starting Point-Teaching Entry Level Geoscience 2013). Uma prova disso é a seguinte tabela que resume uma comparação entre o GBL e o ensino tradicional.

Tabela 2 GBL vs Ensino tradicional (New Media Institute 2013)

	Traditional Training (lectures, online tutorials)	Hands-on Training	Game-based Learning
Cost-effective	X		X
Low physical risk/liability	X		X
Standardized assessments allowing student-to-student comparisons	X		X
Highly engaging		X	X
Learning pace tailored to individual student		X	X
Immediate feedback in response to student mistakes		X	X
Student can easily transfer learning to real-world environment		X	X
Learner is actively engaged		X	X

Facilmente se percebe através da Tabela 2 que o GBL consegue combinar as vantagens do estudo teórico e treino prático, duas atividades do estudo tradicional. Com isto se pode concluir que, quando bem desenvolvido, um GBL tem um valor acrescido sobre o estudo tradicional pois para além das vantagens a nível educativo tem a vantagem de ser um jogo e tornar o estudo numa atividade divertida.

2.5. Engenharia de *Software*

O problema desta dissertação está centrado na disciplina de Engenharia de *Software*. Esta disciplina é uma área da computação focada na definição, desenvolvimento e manutenção de sistemas de *software*, visando a organização, a produtividade e a qualidade do produto com a aplicação de tecnologias e práticas de outras disciplinas.

Segundo a última versão oficial do SWEBOK (Bourque et al. 2004) um documento criado pelo IEEE com a finalidade de servir de referência na área de Engenharia de *Software*, esta disciplina é composta por um vasto conjunto de áreas de conhecimento. Na Ilustração 1 são apresentadas as áreas de conhecimento que integram a Engenharia de *Software*.

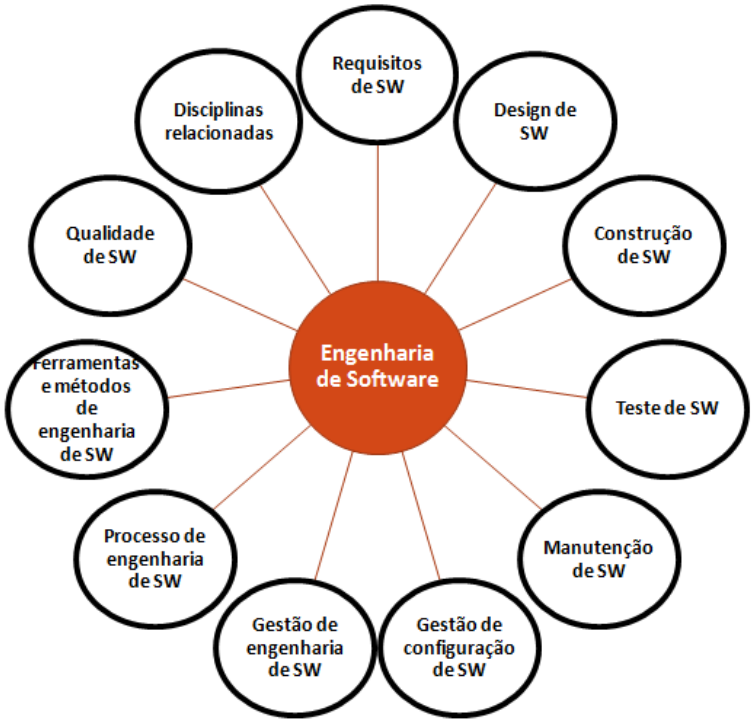

Ilustração 1 Áreas de conhecimento da Engenharia de *Software*

Como a Engenharia de *Software* é uma disciplina bastante vasta, decidiu-se considerar apenas parte dela para o desenvolvimento deste projeto. A área de conhecimento escolhida, por ser das menos exploradas, foi então a Qualidade de *Software*, que engloba atividades como verificação e validação de *software*.

2.5.1. Verificação & Validação de Software

Verificação e Validação são processos independentes utilizados na área de Engenharia de *Software*, que são utilizados em conjunto para garantir que um produto, serviço ou sistema cumprem os requisitos e especificações definidas e que vão de encontro ao propósito do seu desenvolvimento.

Verificação é o processo de avaliação de um sistema ou componente para determinar se o produto satisfaz as especificações determinadas no início da fase. Ou seja, "Estamos a construir o produto corretamente?".

Validação é o processo de avaliação para determinar se o sistema atende às necessidades e requisitos dos utilizadores. Ou seja, "Estamos a construir o produto correto?" (Sommerville 2011; Software Testing Fundamentals 2013).

2.5.2. Revisão de Código

É uma tarefa que integra a área de conhecimento Qualidade de *Software* que consiste na análise de código fonte na tentativa de encontrar e corrigir os erros que tenham escapado na fase

inicial de desenvolvimento, melhorando tanto a qualidade geral do *software* como as habilidades dos programadores (Sommerville 2011).

3. Revisão Bibliográfica

Este capítulo apresenta o trabalho de pesquisa efetuado, nomeadamente, o estudo do estado da arte feito a fim de averiguar a viabilidade do desenvolvimento deste projeto. Foi realizado um estudo na área dos jogos de computador para Engenharia de *Software* já existentes com a finalidade de avaliar as áreas de conhecimento menos exploradas, procurando desenvolver uma solução numa área onde a concorrência fosse menos forte.

3.1. Introdução

Como anteriormente foi referido, os jogos sérios para o ensino de Engenharia de *Software* não estão ainda muito explorados e o que existe insere-se, sobretudo, apenas em algumas das áreas de conhecimento. Por isso, a possibilidade de criar uma solução para a Engenharia de *Software*, principalmente numa das áreas de conhecimento menos exploradas, que traga vantagem ao utilizador pela sua utilização como acontece com os simuladores de voo por exemplo, é a grande motivação para o desenvolvimento deste projeto.

A indústria dos jogos de computador é uma das mais rentáveis em todo o mundo e a oferta é cada vez maior. Atualmente existe uma quantidade considerável de jogos sérios e a área de Engenharia de *Software* também conta com alguns exemplares bem sucedidos.

A maior parte dos jogos sérios para Engenharia de *Software* insere-se sobretudo em áreas de conhecimento como Construção de *Software*, Processo de Engenharia de *Software* e Gestão de Engenharia de *Software*. Segundo a pesquisa efetuada, não há conhecimento de jogos desenvolvidos na área de Qualidade de *Software*, mais propriamente, que abordem a Revisão de Código, área na qual se insere a aplicação desenvolvida. No entanto, de seguida, apresentam-se alguns exemplos de jogos sérios para Engenharia de *Software* (Game-Based Learning Dev 2013).

3.2. Jogos Sérios de Programação

Robocode: é um jogo *open source* de aprendizagem projetado para ajudar as pessoas a aprenderem a programar em Java e desfrutar da experiência. O jogo consiste em programar um robô seguindo uma API para depois o introduzir num campo de batalha com outros robôs de forma a testar a eficácia da inteligência artificial do mesmo (Robocode 2013).

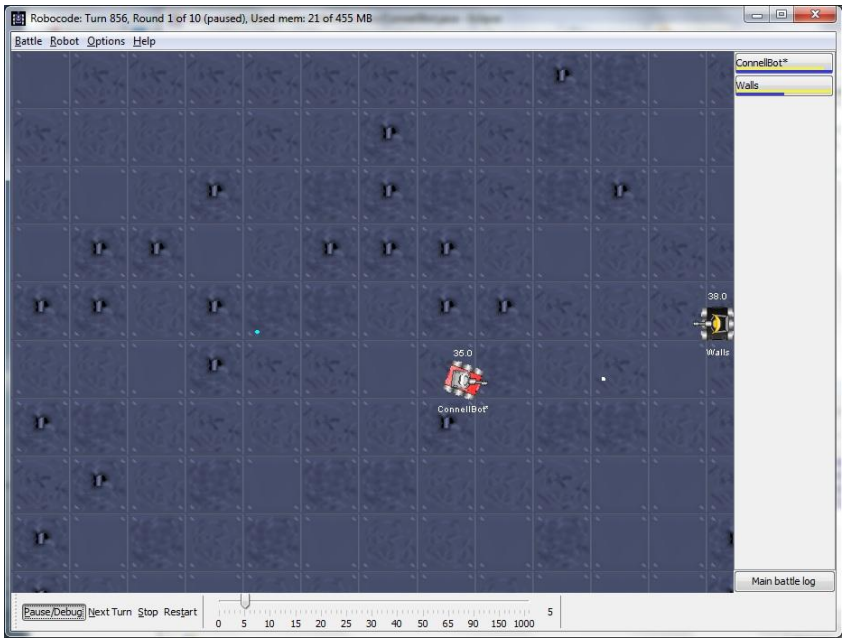

Ilustração 2 Robocode - Ambiente de jogo

Alice: é um *software* educativo utilizado como uma ferramenta de ensino para computação introdutória que ensina os alunos a programar em ambiente 3D (Alice 2013).

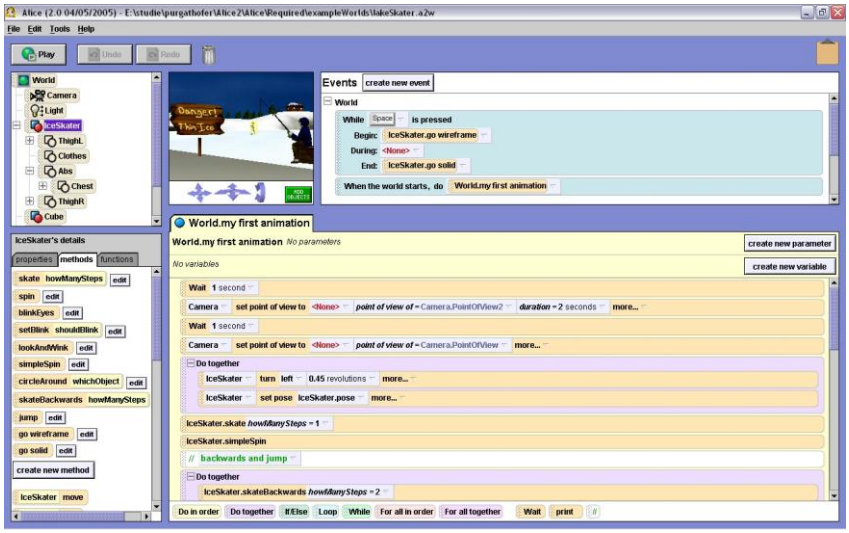

Ilustração 3 Alice - Ambiente de jogo

3.3. Jogos Sérios de Processo e Gestão de Engenharia de *Software*

SimSE: é um simulador de processos de Engenharia de *Software* onde o papel do jogador é ser gestor de um projeto. O objetivo do jogo é gerir um projeto da melhor forma possível escolhendo as tarefas para os funcionários consoante grau de experiência e fadiga. O jogo começa com um orçamento e um tempo estimado para o projeto. Este jogo tem como finalidade estabelecer uma ligação entre a grande quantidade de conhecimento conceptual dado aos estudantes nas aulas e a pequena quantidade desse conhecimento aplicada em casos práticos (SimSE 2013).

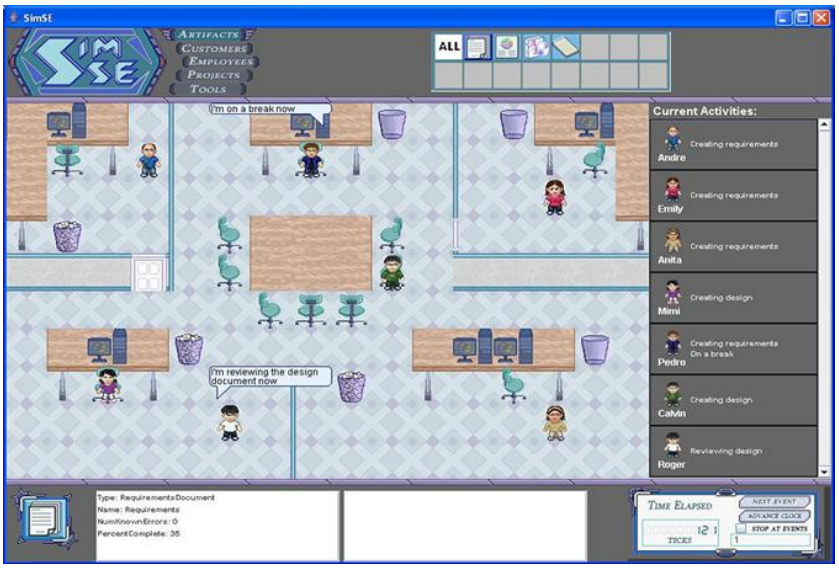

Ilustração 4 SimSE - Ambiente de jogo

MO-SEProcess: é um jogo multijogador 3D de processo de Engenharia de *Software* baseado no jogo SimSE desenvolvido para o Second Life. O jogo tem como objetivo ensinar os princípios do processo de engenharia de *software*, simulando o desenvolvimento de um projeto de *software* de tamanho moderado (MO-SEProcess 2013).

Ilustração 5 MO-SEProcess - Ambiente de jogo

3.4. Conclusão

O estado da arte relativo aos jogos sérios para Engenharia de *Software*, como referido anteriormente, conta já com alguns bons exemplares. Acima podemos ver 4 exemplos que abrangem 3 áreas de conhecimento diferentes, Construção de *Software*, Processo de engenharia de *Software* e Gestão de engenharia de *Software*. Estas são as áreas de Engenharia de *Software* que mais estão exploradas a nível de jogos sérios. Quanto a jogos que abranjam a área de conhecimento abordada neste projeto não há conhecimento de nenhum exemplar. Por isso, é já uma mais-valia para este projeto uma vez que não terá concorrência direta nessa área de conhecimento.

4. Definição do Conceito do Jogo

O presente capítulo apresenta uma descrição detalhada do trabalho de dissertação desenvolvido. Além da seleção do tema, são apresentados também os objetivos, tanto do jogo como de aprendizagem, e ainda o conjunto de funcionalidades que a solução desenvolvida integra.

4.1. Seleção do Tema

O presente trabalho de dissertação tem como problema a resolver o desenvolvimento de um jogo sério a ser aplicado no ensino de Engenharia de *Software*. Após a pesquisa efetuada relativamente ao estado da arte, a opção escolhida para solução ao problema recaiu sobre as áreas de conhecimento da Engenharia de *Software* menos exploradas pelos jogos sérios. Como fora referido anteriormente neste documento, a solução desenvolvida consiste num jogo que permite praticar técnicas de Revisão de Código, atividade essa que faz parte da Qualidade de *Software*, uma das áreas de conhecimento da Engenharia de *Software* menos exploradas pelos jogos sérios.

4.2. Objetivo do Jogo

A aplicação tem como objetivo de jogo, comum a todos os jogos, obter melhores pontuações que os outros utilizadores. Para isso é preciso ser-se eficaz, encontrando o maior número de erros sem assinalar erros inexistentes (falsos positivos), e eficiente, executando essa tarefa no menor espaço de tempo possível. A pontuação baseia-se então nestas três componentes: erros corretos, erros errados e tempo gasto. A pontuação é dada sob a forma de percentagem para ser possível comparar exercícios de diferente escala.

4.3. Objetivo de Aprendizagem

Com esta aplicação os utilizadores têm a possibilidade de praticarem técnicas de revisão de código de uma forma mais agradável do que com os métodos de estudo tradicionais. Para além disso torna o estudo menos maçador por ser mais simples e mais rápido do que a tradicional folha de papel para resolução deste tipo de problemas. Funciona um bocado à semelhança dos testes de exame de código. Como todos sabemos, existem exames em formato impresso e em formato digital sendo que o formato digital tem muito mais sucesso por várias razões: não ocupa espaço, não tem gastos em papel e acima de tudo apresenta o resultado obtido de forma automática sem precisar que o utilizador vá confirmar por si mesmo perdendo tempo de estudo precioso.

4.4. Funcionalidades

A solução desenvolvida consiste numa aplicação que permite praticar técnicas de Revisão de Código como anteriormente foi dito. A aplicação dispõe de três grandes funcionalidades, carregar exercício, resolver exercício e consultar pontuações como se pode ver na Ilustração 6 que apresenta o diagrama de casos de uso.

Quanto à funcionalidade de carregamento de exercícios, os utilizadores, no papel de professores/autores, apenas têm de escolher o ficheiro que pretendem carregar e posteriormente simular um jogo definindo os erros para servirem como solução do exercício. Durante a simulação de jogo o utilizador pode definir uma *checklist* predefinida ou uma personalizada para classificar os tipos de erros do exercício. Nesta segunda opção o utilizador adiciona novos itens à *checklist* à medida que vá precisando. A *checklist* indica o conjunto de vários itens a verificar como por exemplo, comentários de documentação, adesão a convenções de nomes, validação de pré condições, validação de pós condições, entre outros. O facto desta aplicação se encontrar ligada a um servidor, permite a partilha de exercícios entre os utilizadores.

Relativamente à funcionalidade de resolução de exercícios, os utilizadores, no papel de alunos/jogadores, iniciam o jogo escolhendo o exercício que pretendem resolver. Este, quando em modo de jogo, deparando-se com um conjunto de linhas de código, deve apenas identificar todos os erros que encontrar com base numa *checklist*. A marcação de um erro é composta pela seleção do segmento de código pretendido e pela indicação do tipo de erro segundo a *checklist* que é apresentada com o clique do botão direito do rato. Após dar o jogo por terminado ser-lhe-á apresentado um conjunto de estatísticas com os resultados obtidos assim como uma pontuação de forma percentual com base nos erros identificados e no tempo gasto para o fazer. O utilizador pode ainda através de dois botões indicar se gostou ou não do exercício para contribuir para as estatísticas do ficheiro.

A outra funcionalidade consiste na consulta de pontuações máximas dos utilizadores. A pontuação de um jogo é calculada com base no número de erros bem e mal assinalados assim como no tempo gasto para resolver o exercício. O resultado é divido por uma pontuação

considerada perfeita para transformá-lo em formato de percentagem. Apesar de a fórmula permitir valores fora dos limites a pontuação obtida nunca é inferior a 0 nem superior a 100. A fórmula da pontuação é então a seguinte:

$$P = \frac{10a - 2b - t}{10N - TN} * 100 <=> \frac{10a - 2b - t}{8N} * 100, \qquad P \in [0,100]$$

P = pontuação obtida (%);

N = número de erros na solução;

T = tempo esperado por erro (estabelecido como 2 minutos);

a = número de erros assinalados corretamente;

b = número de erros assinalados erradamente;

t = tempo total gasto (minutos);

É importante referir que o método racional mais adequado para se chegar a uma fórmula para a pontuação seria baseado numa análise económica tendo em atenção o custo de hora do revisor, o custo da identificação de falsos positivos e o custo de erros não assinalados, mas os parâmetros são extremamente varáveis, pelo que se optou por valores que se julgaram razoáveis, podendo estes serem configurados.

A tabela de pontuações máximas é composta por três colunas: Username, Average Score, Overall Score sendo ordenada por ordem decrescente pela terceira coluna. A coluna Average Score é obtida a partir da média das primeiras pontuações obtidas em todos os ficheiros resolvidos pelo utilizador. A coluna Overall Score é calculada a partir da segunda coluna mas tem também em conta o número de ficheiros já resolvidos pelo utilizador para que quanto maior o número de ficheiros resolvidos melhor seja a pontuação. Eis a fórmula de cálculo:

$$O = \frac{\log(n+1)}{\log(N+1)} * A, \qquad O \in [0,100]$$

O = Overall Score;

A = Average Score;

n = número de ficheiros resolvidos pelo utilizador;

N = número total de ficheiros no sistema;

Este sistema de pontuação funciona como um estímulo para o utilizador fazer mais exercícios para subir na tabela classificativa e previne que um utilizador com apenas um ficheiro resolvido com boa pontuação fique à frente de outro com uma pontuação ligeiramente inferior mas com um número mais elevado de ficheiros resolvidos por exemplo. A utilização de uma escala logarítmica em vez de linear permite que um utilizador possa atingir uma classificação relativamente alta resolvendo uma fração de exercícios mais pequena.

A Ilustração 6 apresenta o diagrama de casos de uso da aplicação.

Ilustração 6 Diagrama de casos de uso

5. Desenho e Implementação

Neste capítulo são apresentados todos os detalhes de implementação dispostos por cinco subcapítulos. O primeiro deles é composto por uma explicação das escolhas tecnológicas, segue-se depois o subcapítulo que apresenta a arquitetura do projeto. Depois, seguem-se então os subcapítulos que descrevem as partes integrantes do projeto sendo eles o Servidor, a Base de Dados e o Cliente, terminando com dois subcapítulo que explicam a forma como as partes comunicam.

5.1. Escolha de Tecnologias

Para o desenvolvimento de qualquer jogo, é comum utilizar-se um motor de jogo como os famosos CryEngine, UDK, Unity3D entre outros. Como a aplicação a desenvolver tem o propósito de um jogo, seria de esperar que se utilizasse um desses motores para a desenvolver. Acontece que estes motores de jogo são mais indicados para jogos com componentes de física como movimento, colisões, gravidade e outros. Uma vez que esta aplicação não carece desse tipo de componentes deixa de ser necessária a utilização deste tipo de motores de jogo. A tecnologia escolhida para o desenvolvimento desta aplicação consiste numa aplicação Java que atua como cliente e conecta por *web services* com um servidor desenvolvido também em Java que por sua vez se conecta com uma base de dados em MySQL, um dos sistemas de gestão de base de dados *open source* mais popular. A portabilidade, a possibilidade de migração para Java Applet podendo assim ser integrado em página web, o nível de interatividade permitido, o grau de conhecimento da linguagem e ainda a popularidade foram alguns dos fatores que influenciaram a escolha de uma aplicação Java para a solução deste trabalho. Quanto à arquitetura da solução consiste num sistema distribuído cliente-servidor capaz de funcionar sobre a Internet que tem em atenção algumas questões de segurança como por exemplo o facto

de o cliente não aceder diretamente à base de dados, apenas com invocação ao servidor. Tanto o cliente como o servidor foram desenvolvidos no IDE Eclipse.

No desenvolvimento deste projeto foram utilizadas ainda várias APIs que serão identificadas ao longo do capítulo 5 na secção mais indicada.

5.2. Arquitetura

O produto desenvolvido é composto por um conjunto de 3 partes, sendo elas, o servidor, a base de dados e o cliente. Estas 3 partes serão descritas em maior detalhe nos subcapítulos seguintes. Antes de passar à explicação detalhada eis a Ilustração 7 que mostra um diagrama de *deployment* que representa a arquitetura do produto.

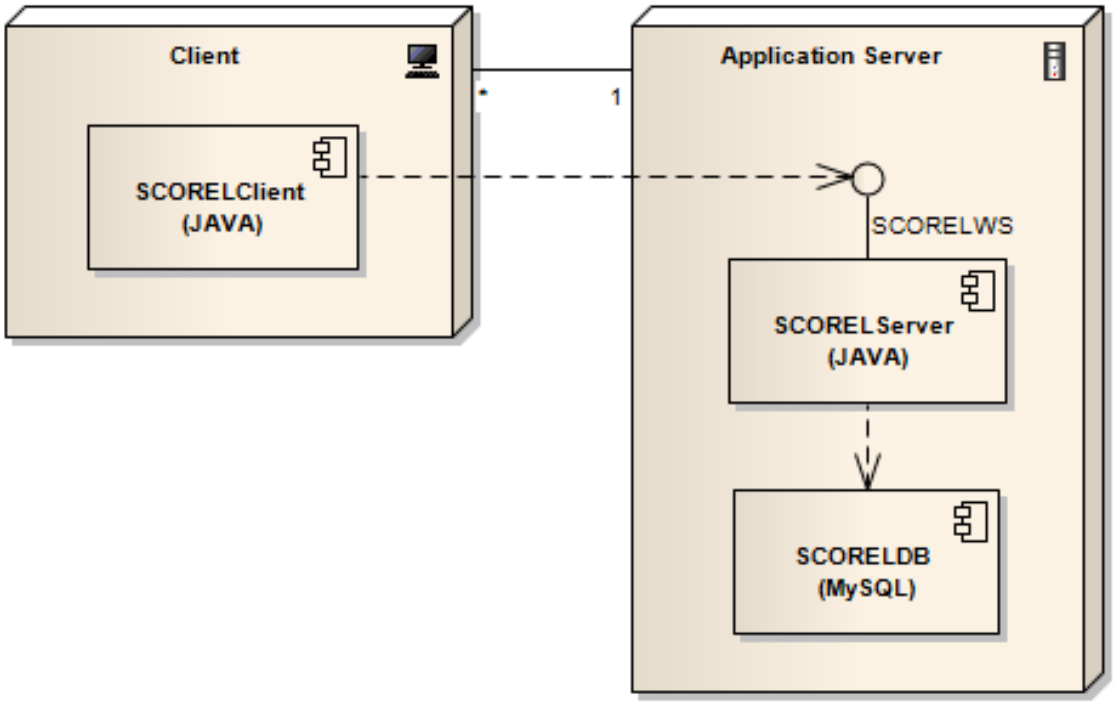

Ilustração 7 Diagrama de arquitetura

5.3. Servidor

O servidor desta aplicação baseia-se num serviço web do tipo JAX-WS Web Service ou seja, um serviço web baseado em XML. Este é implementado de forma autónoma, ou seja, apenas é utilizada uma aplicação Java que publica um serviço web através do método *publish* da classe *javax.xml.ws.Endpoint*. Este comando cria um ambiente de execução de serviços web incluído no Java SE 6 e implanta-o num servidor leve HTTP. O serviço é depois acedido pelos clientes através do endereço fornecido no método *publish* que disponibiliza um ficheiro em formato WSDL, uma linguagem baseada em XML para descrever serviços web. No capítulo 5.6 este tema será explicado em maior detalhe enquadrando-se na explicação da comunicação

cliente-servidor. O servidor dispõe ainda de uma comunicação com uma base de dados MySQL que será descrita com maior detalhe no capítulo 5.7.

5.4. Base de Dados

A base de dados, desenvolvida em MySQL, conta com um conjunto de sete tabelas necessárias para garantir o funcionamento da aplicação. A Tabela 3 apresenta a lista de tabelas da base de dados e respetiva descrição. A Ilustração 8 representa o diagrama relacional da base de dados.

Tabela 3 Tabelas da base de dados

Nome	Descrição
user	Tabela onde são guardados os registos dos utilizadores.
hile	Tabela onde são guardados os ficheiros para os exercícios.
error	Tabela onde são guardados os erros dos ficheiros.
score	Tabela onde são guardados os primeiros jogos dos utilizadores em cada ficheiro.
likes	Tabela onde são guardados os *likes* dos utilizadores nos ficheiros.
dislikes	Tabela onde são guardados os *dislikes* dos utilizadores nos ficheiros.
types	Tabela onde são guardadas as *checklists* dos ficheiros.

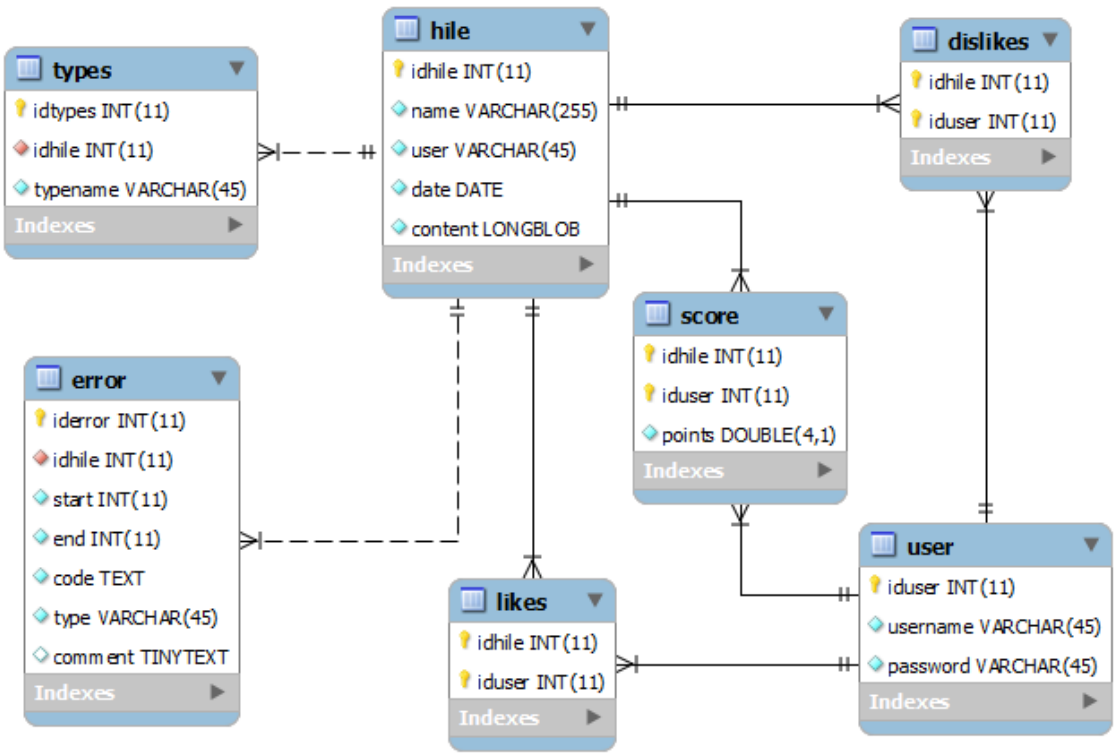

Ilustração 8 Diagrama relacional da base de dados

5.5. Cliente

O cliente é uma aplicação Java com uma interface gráfica desenvolvida através da biblioteca SWING. O cliente encontra-se divido em três pacotes, um pacote principal relativo ao funcionamento da aplicação (pt.up.fe.diss), um pacote de interface gráfica (pt.up.fe.diss.gui) e um último pacote relativo à comunicação com o serviço web disponibilizado pelo servidor (pt.up.fe.diss.jaxws). No capítulo 5.6 será explicado em detalhe como é feita esta comunicação. A Ilustração 9 apresenta o diagrama UML do projeto do cliente.

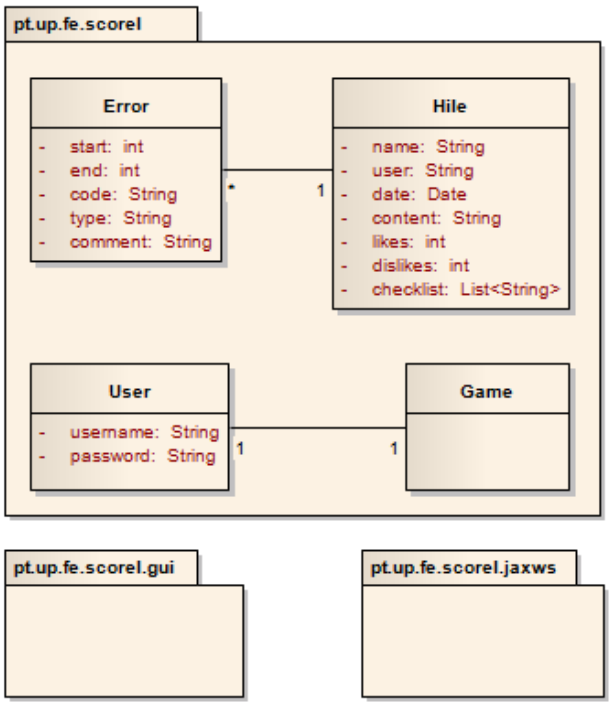

Ilustração 9 Diagrama UML do cliente

A interface gráfica do cliente foi desenvolvida através da biblioteca SWING com o auxílio do WindowBuilder, um *plugin* para Eclipse que permite desenvolver o ambiente gráfico através de ferramentas WYSIWYG. O desenvolvimento da interface gráfica baseia-se num conjunto de painéis, compostos por vários componentes, que vão transitando entre si. De entre os componentes utilizados destacam-se o JLable (área de texto não editável), o JTestArea (área de texto editável), o JButton (botão que ativa uma ação) e o JTextPane (área de texto personalizável). Seguem-se algumas ilustrações dos painéis principais da aplicação com os respetivos componentes assinalados.

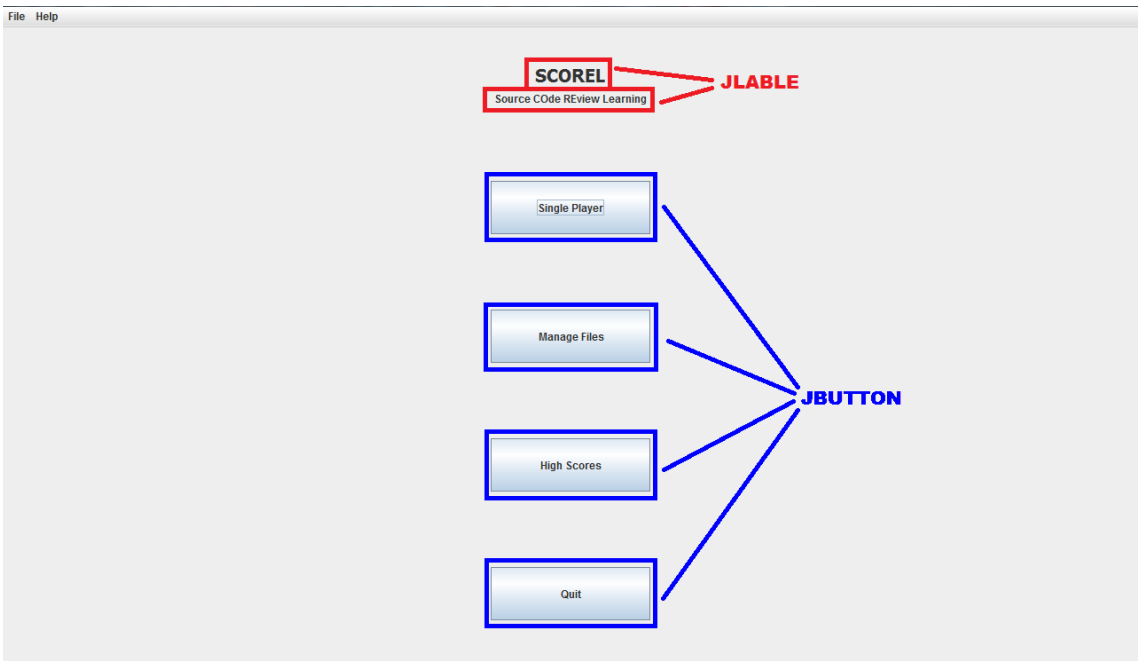

Ilustração 10 Painel do menu principal

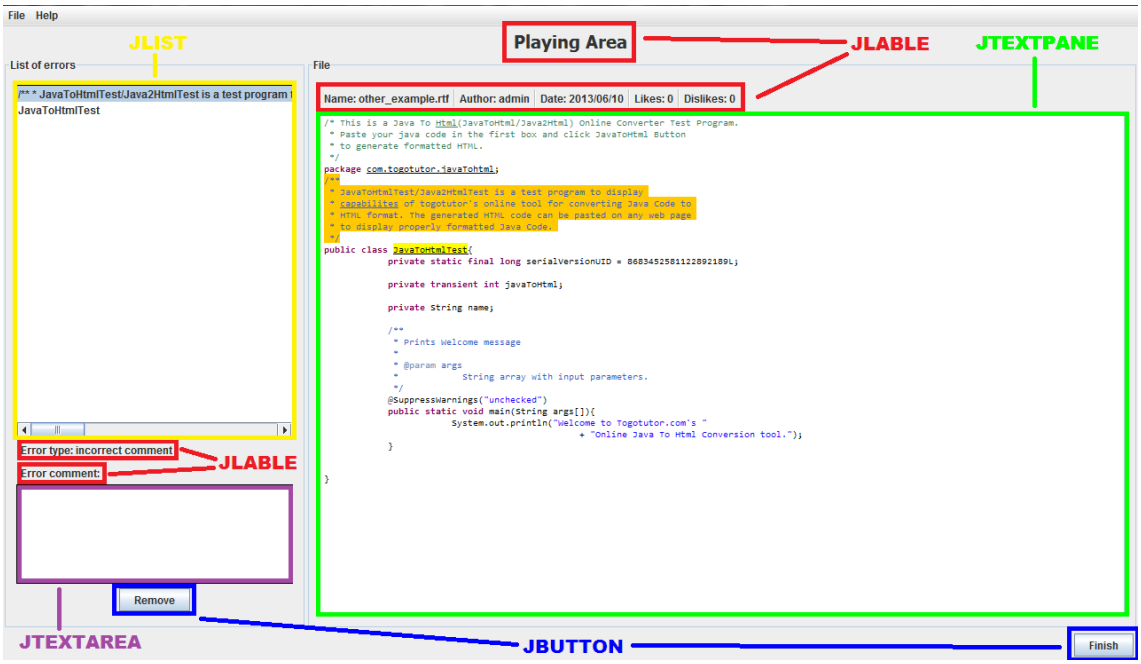

Ilustração 11 Painel da área de jogo

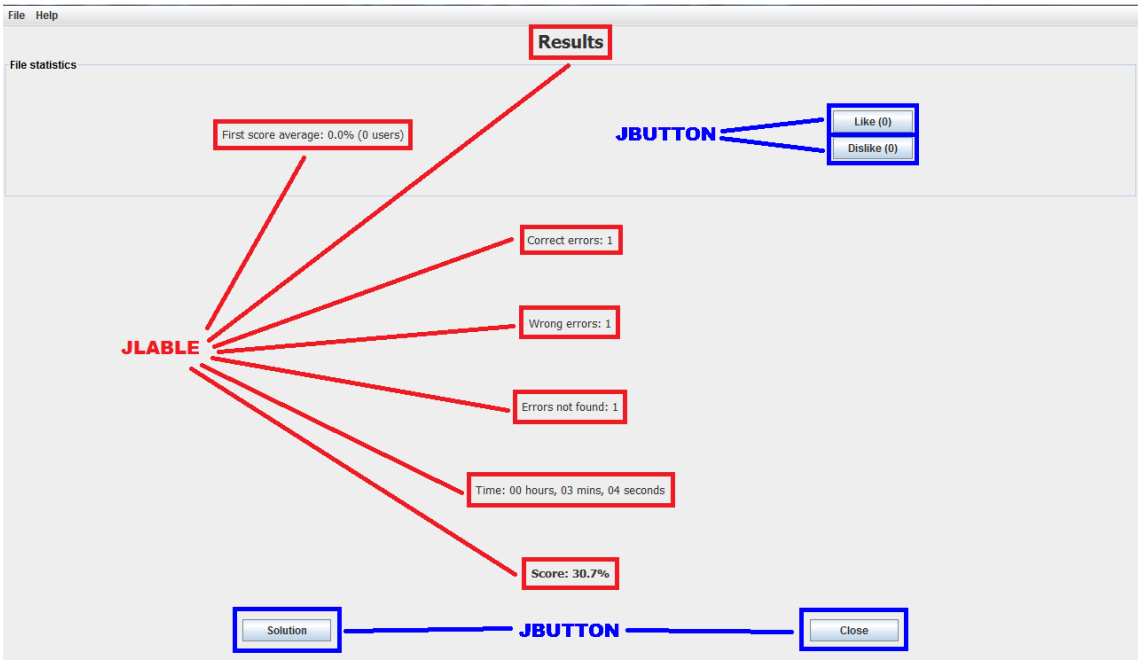

Ilustração 12 Painel do fim do jogo

Uma vez que a aplicação é composta por um conjunto de painéis, é importante perceber como funciona a transição entre eles. Eis então a Ilustração 13 que apresenta o diagrama de estados da aplicação, sendo cada estado, a representação de um painel da aplicação e a etiqueta de transição, o nome do botão que origina tal transição.

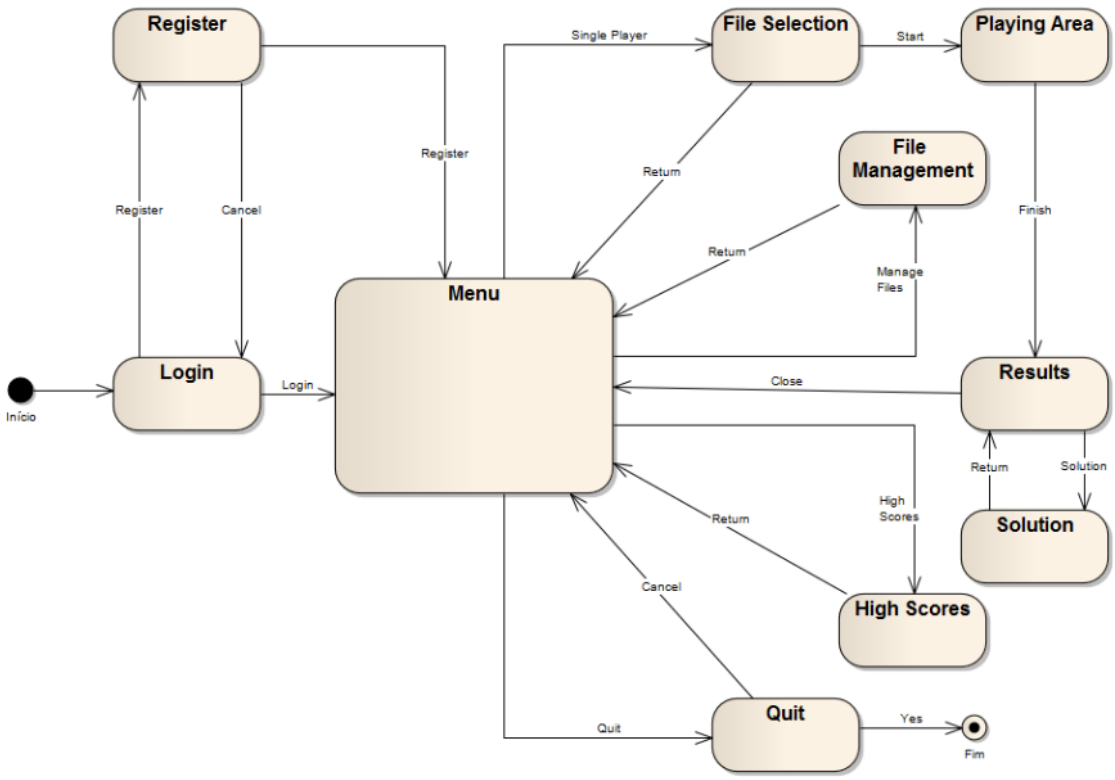

Ilustração 13 Diagrama de navegação entre painéis

A Ilustração 13 apenas representa a transição entre painéis a partir dos componentes dos mesmos. Há também um conjunto de transições entre painéis, ainda que menor, a partir da barra de menus da janela, que foram omitidos para facilitar a interpretação do diagrama de estados.

5.6. Comunicação Cliente – Servidor

A comunicação entre o cliente e o servidor é feita com base na biblioteca JAX-WS, uma biblioteca de chamadas remotas, em inglês, RPC - Remote Procedure Call. O objetivo do RPC é permitir que o cliente faça invocações de funções do servidor como se fossem funções locais. Todos os aspetos de comunicação e de representação tratados pela biblioteca RPC e por objetos gerados automaticamente designados por *ties*, no lado do servidor e por *stubs* no lado do cliente. A Ilustração 14 apresenta em detalhe a estrutura de um JAX-WS Web Service.

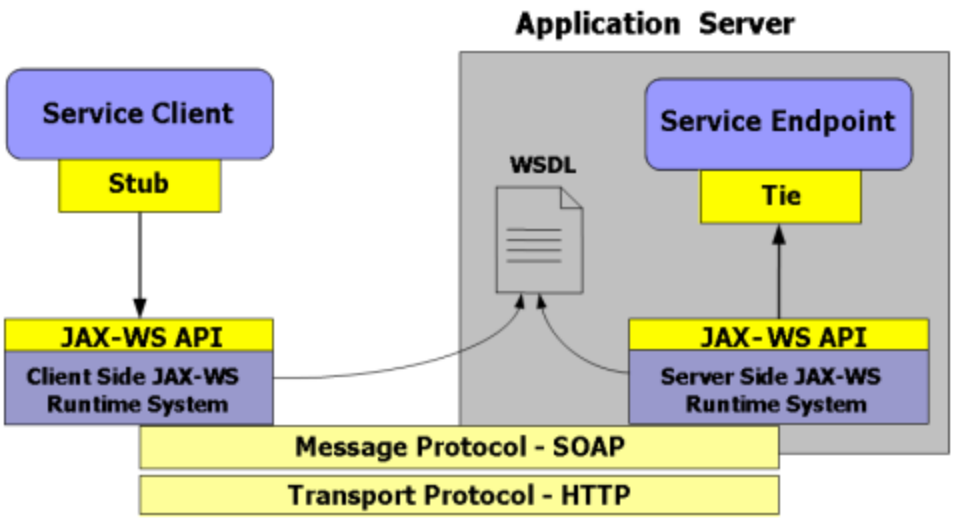

Ilustração 14 Estrutura do JAX-WS (Docentes de Sistemas Distribuídos, DEI, IST 2013)

O desenvolvimento de um serviço deste género é feito de uma forma bastante simplificada. Primeiramente foi necessário desenvolver as classes do serviço, anotando-as segundo a biblioteca *javax.jws* e de seguida desenvolver uma classe que publicasse esse serviço. Eis como fazê-lo abaixo:

WS.java (Endpoint Interface)

```
@WebService(name="WS", serviceName = "WSService", portName = "WSPort")
public interface WS {
	@WebMethod
	boolean checkConnection();
	...
}
```

WSImpl.java (Endpoint Implementation)

```
@WebService(endpointInterface = "PACKAGENAME.WS")
public class WSImpl implements WS{
```

```
        @Override
        boolean checkConnection(){...}
        ...
}
```

WSPublisher.java (Endpoint Publisher)

```
public class WSPublisher {
        public static void main(String[] args){
                String address = "http://localhost:8080/WS";
                WSImpl service = new WSImpl();
                Endpoint.publish(address, service);
        }
}
```

Posteriormente foi necessário gerar os *stubs* para o cliente poder comunicar com o serviço. Estes foram gerados automaticamente com o comando wsimport da framework JAX-WS RI a partir do WSDL disponível no endereço onde o serviço é publicado através do Enpoint Publisher. O comando é o seguinte: wsimport -keep -verbose http://localhost:8080/WS?wsdl.

Por fim, incluindo os *stubs* gerados no projeto do cliente, criou-se o meio de conexão com o serviço para poder invocar as funções que se desejar. Eis como fazê-lo abaixo:

```
        WSImplService service = new WSImplService();
        WS wsPort = service.getWSImplPort();
        wsPort.checkConnection();
```

Outra biblioteca útil para a comunicação cliente-servidor foi a JAXB. Esta biblioteca é utilizada para permitir o retorno de tipos complexos nas funções do serviço. Para isso é necessário criar a classe do tipo complexo no projeto do servidor e anotá-la segundo a biblioteca *javax.xml.bind.annotation*. Eis como fazê-lo abaixo:

ComplexData.java

```
@XmlRootElement(name = "complexData")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ComplexDataType", propOrder = {
                "attribute1",
                "_elements"
})
public class ComplexData {
        @XmlElement(name = " attribute1")
        private String attribute1;

        @XmlAnyElement
        private Collection<org.w3c.dom.Element> _elements = null;
        ...
}
```

A geração dos *stubs,* através do comando wsimport acima referido, gera também as classes necessárias para o retorno de objetos do tipo ComplexData. Do lado do cliente objetos deste tipo assumem o nome ComplexDataType.

5.7. Comunicação Servidor – Base de Dados

Como referido na secção 5.4 a base de dados do projeto foi desenvolvida em MySQL. Para estabelecer a comunicação entre o servidor e a base de dados foi necessária a utilização da biblioteca JDBC. Esta biblioteca é utilizada quando se pretende estabelecer a comunicação entre uma aplicação Java e uma base de dados em MySQL. Para estabelecer esta ligação foi necessário instalar o MySQL Workbench, uma ferramenta gráfica para a modelação de bases de dados, que permite criar uma ligação para posteriormente ser acedida pela aplicação Java. A conexão com a base de dados a partir da aplicação Java é feita da seguinte forma:

```
private Connection connection = null;
private String driver = "com.mysql.jdbc.Driver";
private String server = "localhost";
private String dataBaseName = "wsdatabase";
private String username = "user";
private String password = "pass";
private String url = "jdbc:mysql://" + server + ":3306/" + dataBaseName;
...
try {
      Class.forName(driver);
      connection = DriverManager.getConnection(url, username, password);
} catch(ClassNotFoundException ex) {
      System.err.println(ex.getMessage());
} catch(IllegalAccessException ex) {
      System.err.println(ex.getMessage());
} catch(InstantiationException ex) {
      System.err.println(ex.getMessage());
} catch(SQLException ex) {
      System.err.println(ex.getMessage());
}
```

6. Utilização e Experimentação

Este capítulo apresenta o resultado final do trabalho de dissertação. É composto por dois subcapítulos, tendo o primeiro a mesma finalidade de um manual de utilização e sendo o segundo uma demonstração de um de um caso experimental realista utilizado nas aulas de Engenharia de *Software*.

6.1. Modo de Utilização

A aplicação inicia com um painel de *login* onde o utilizador se pode autenticar ou então escolher a opção de registo caso não esteja registado, conforme é apresentado na Ilustração 15.

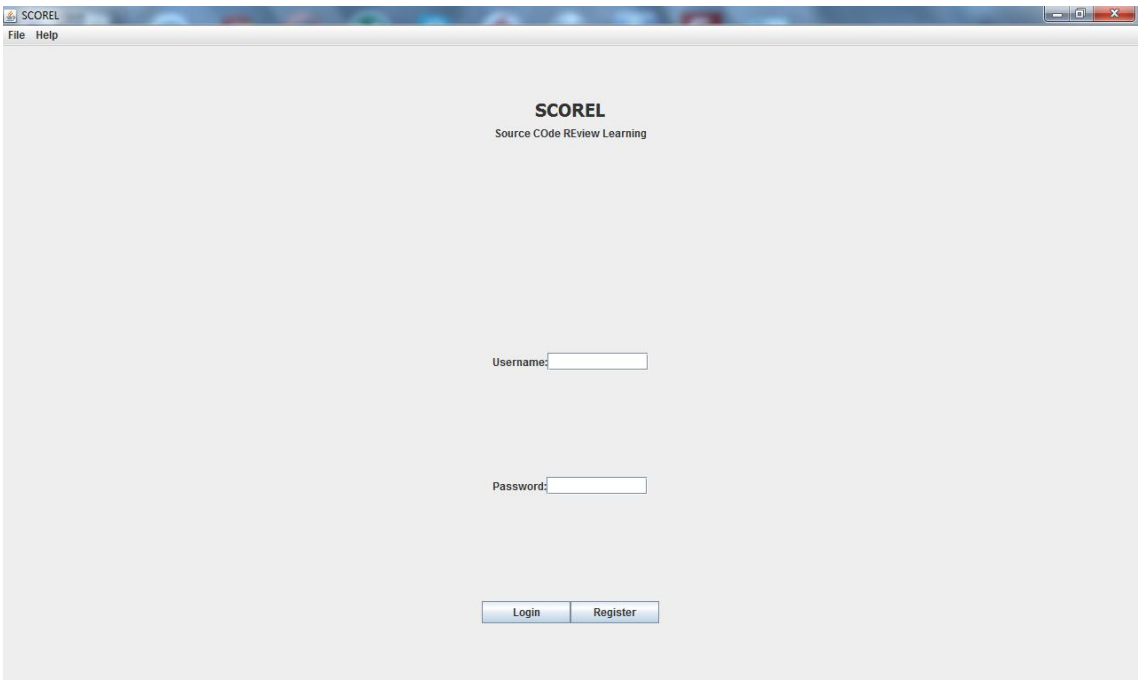

Ilustração 15 Painel de login

No caso de ser um novo utilizador e então optar por fazer novo registo ser-lhe-á apresentado um painel onde deve preencher os campos apresentados e validar o registo tal como mostra a Ilustração 16.

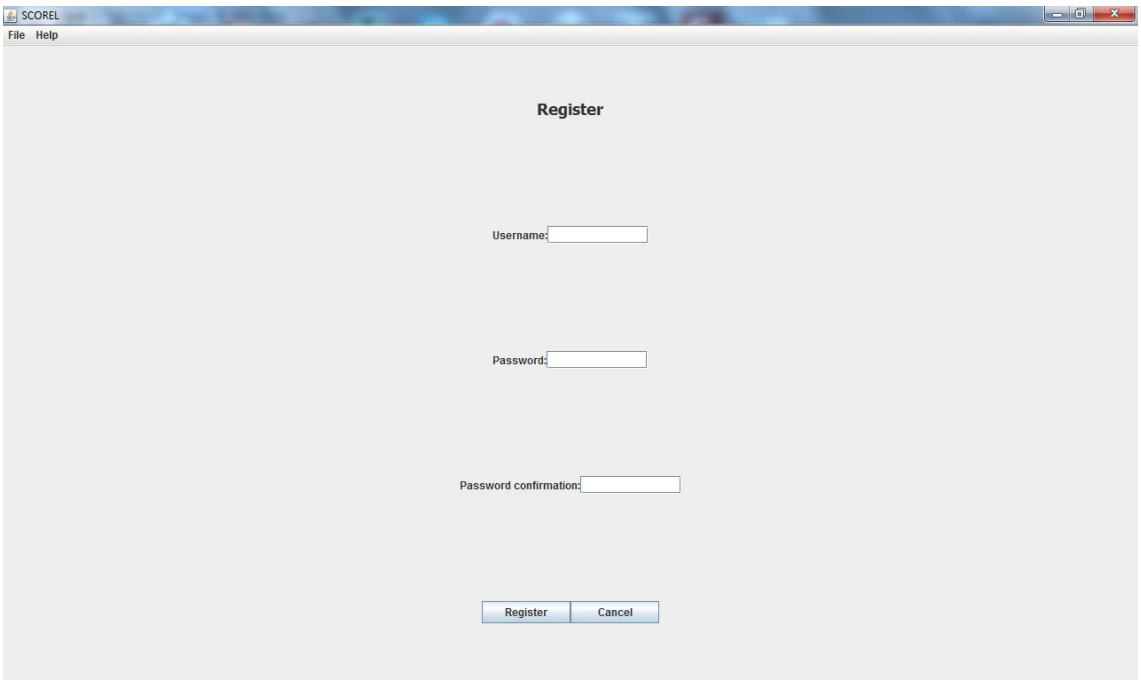

Ilustração 16 Painel de registo

Qualquer uma das duas opções anteriores, login ou registo, leva a que seja apresentado um painel com o menu principal. Uma vez neste painel o utilizador tem à sua disposição um conjunto de botões como é mostrado na Ilustração 17.

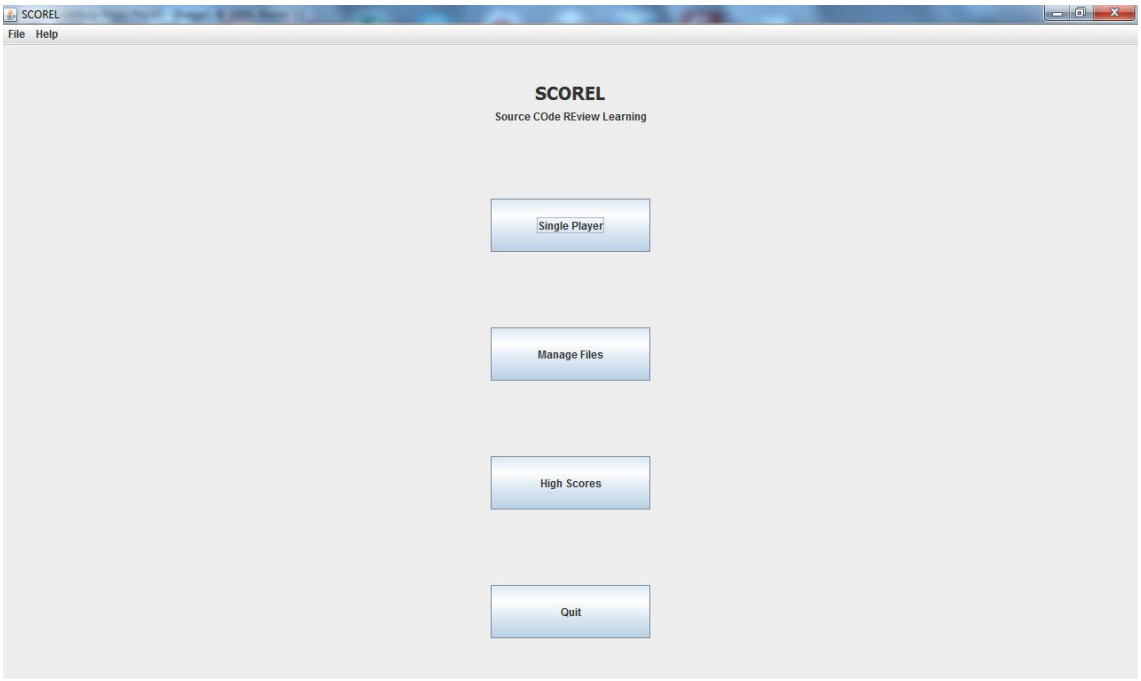

Ilustração 17 Painel do menu principal

Escolhendo a opção "Manage Files" o utilizador depara-se com o painel representado pela Ilustração 18 onde pode gerir os seus ficheiros no sistema eliminando existentes ou carregando novos ficheiros. Neste caso o utilizador ainda não carregou nenhum ficheiro.

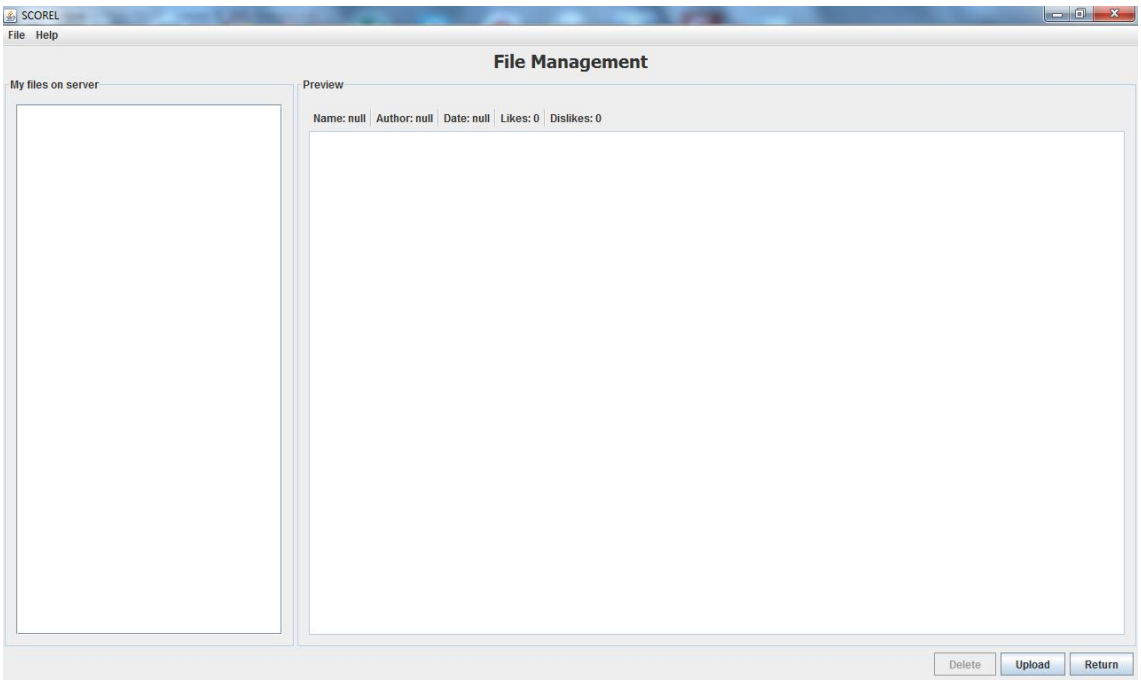

Ilustração 18 Painel de gestão de ficheiros

Escolhendo a opção "Upload" presente no painel da Ilustração 18 surge uma caixa de diálogo que permite ao utilizador selecionar o ficheiro pretendido como mostra a Ilustração 19. Como foi referido noutras secções deste documento, a aplicação suporta o carregamento de ficheiros RTF e HTML.

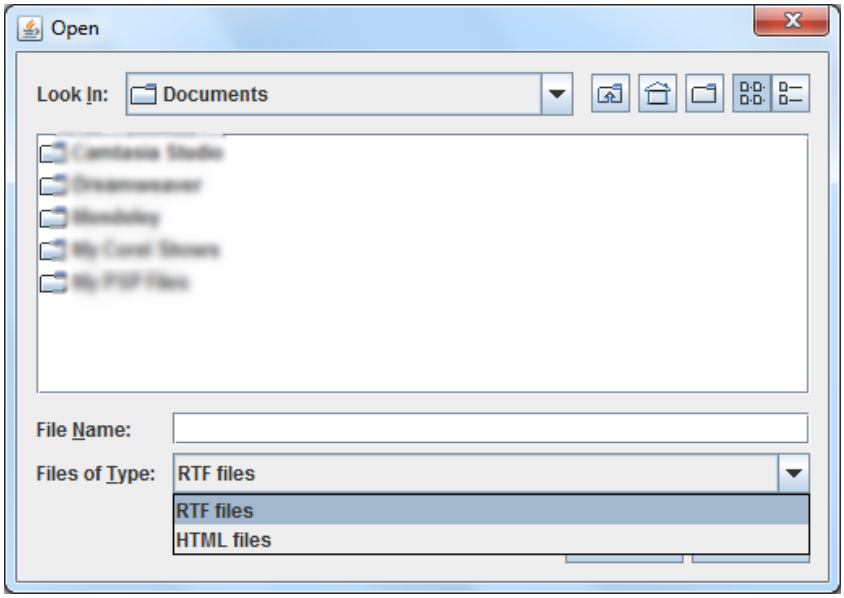

Ilustração 19 Seletor de ficheiros

Após selecionar o ficheiro pretendido e confirmar a operação, é apresentado o painel da Ilustração 20 onde o utilizador deve marcar os erros para que sirvam de solução para o ficheiro

carregado. Para marcar os erros o utilizador apenas tem de selecionar a porção de texto pretendida com o rato e, com o botão direito do mesmo, selecionar o tipo de erro em questão. No painel esquerdo vão surgindo os erros marcados sob a forma de lista podendo ser associado um comentário a cada um deles consoante o que esteja selecionado na lista. Para anular um erro, basta seleciona-lo na lista e clicar na opção "Remove" ou na tecla Delete. Os erros podem ser marcados segundo uma *checklist* predefinida ou configurável. Para definir qual utilizar, é apenas necessário escolher a opção pretendida no canto inferior esquerdo do painel. A alteração de tipo de *checklist* provoca um recomeço da marcação de erros devido à diferença de tipos. Escolhendo a opção de *checklist* predefinida, o menu do botão direito do rato tem o aspeto apresentado na Ilustração 20.

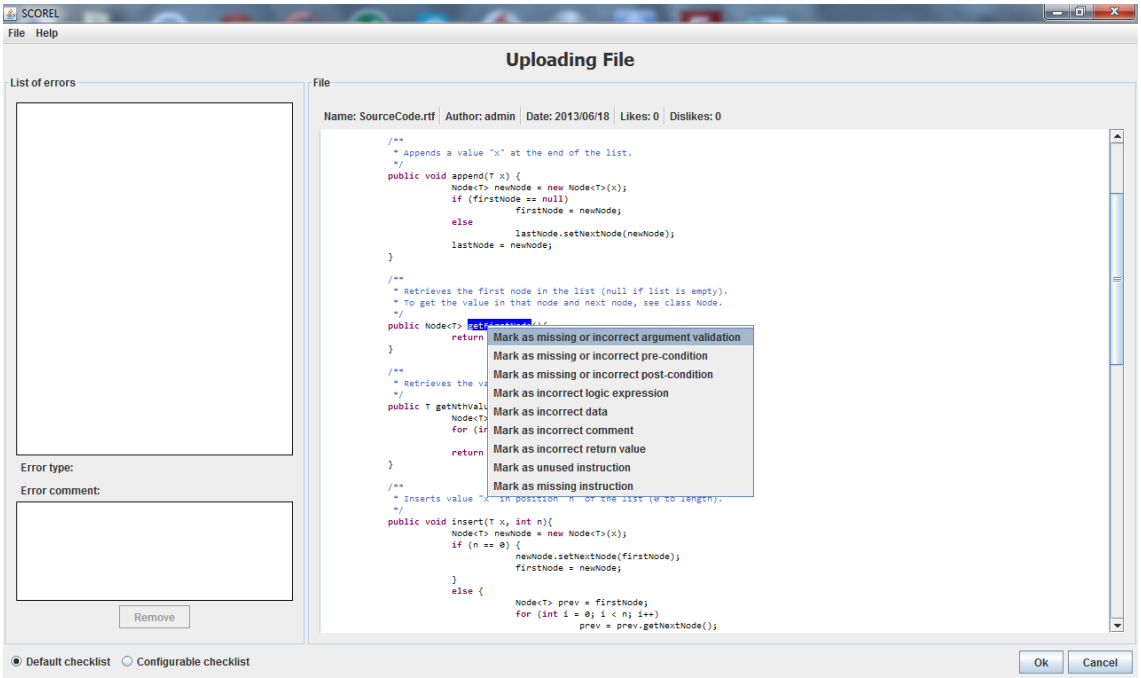

Ilustração 20 Painel de seleção de erros (*checklist* predefinida)

Escolhendo a opção de *checklist* configurável, o menu do botão direito do rato tem o aspeto apresentado na Ilustração 21. O utilizador pode gerir os tipos de erros da *checklist* através dos dois primeiros itens do menu do botão direito do rato. A Ilustração 22 e Ilustração 23 apresentam, respetivamente, as janelas de adição e remoção de tipos da *checklist* configurável.

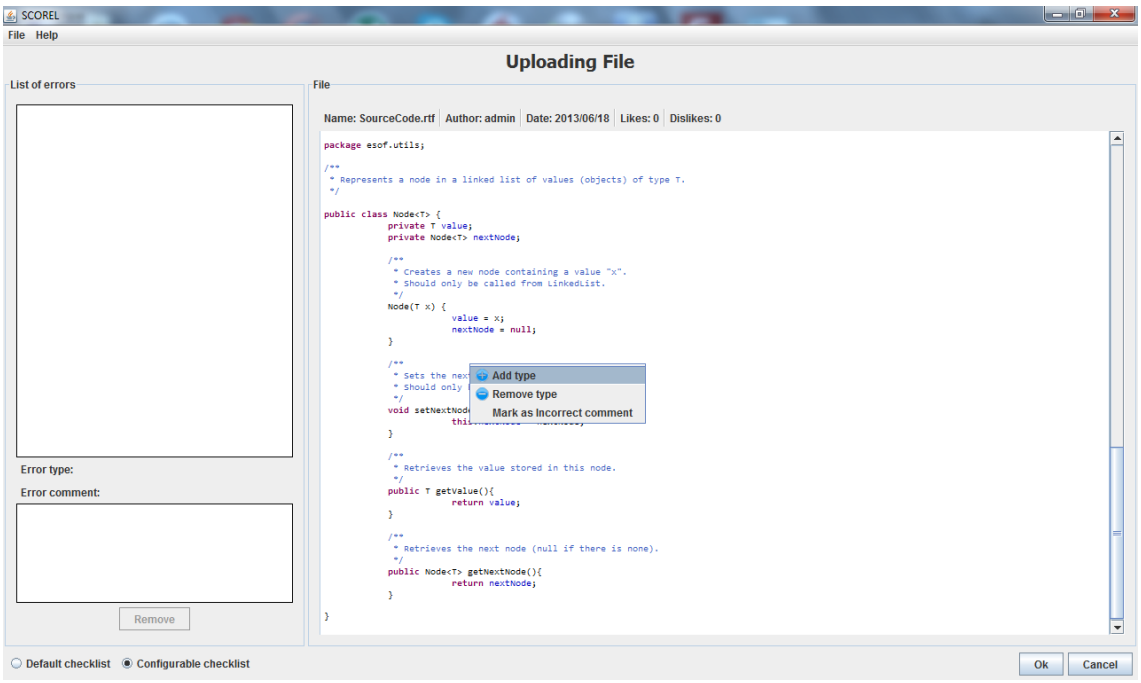

Ilustração 21 Painel de seleção de erros (*checklist* configurável)

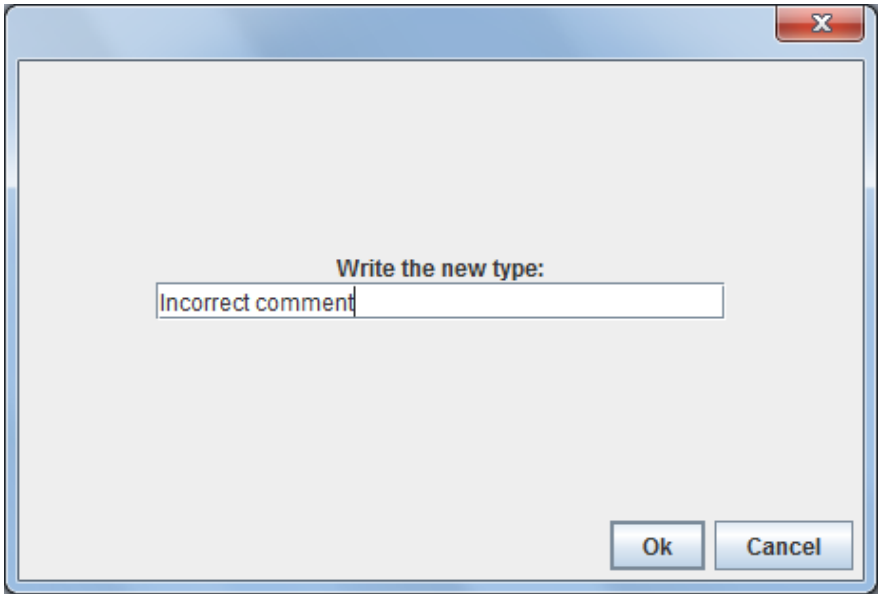

Ilustração 22 Painel de adição de novo tipo

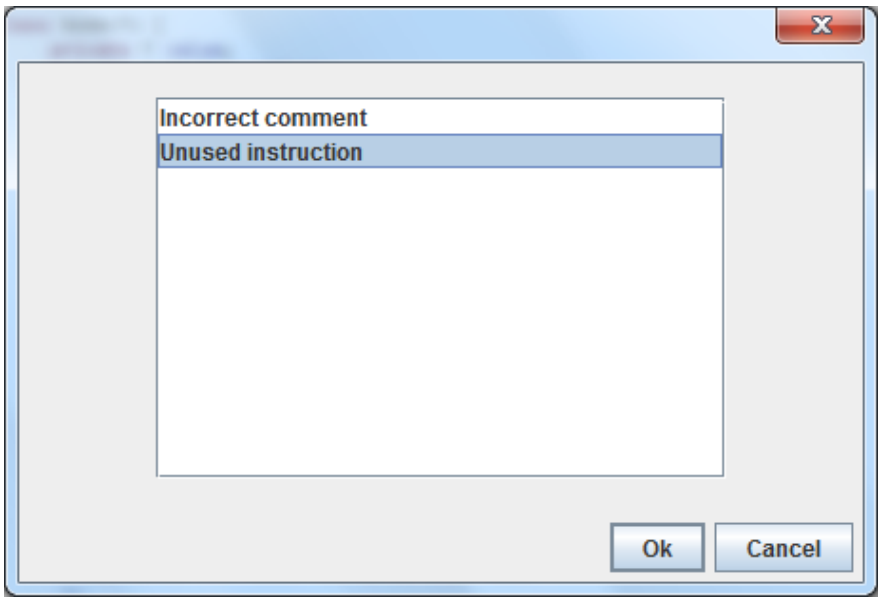

Ilustração 23 Painel de remoção de tipo existente

Havendo ficheiros no servidor, o utilizador pode dar início a um jogo selecionando a opção "Single Player" presente no painel da Ilustração 17. Esta opção conduz a aplicação ao painel presente na Ilustração 24 onde o utilizador tem acesso a uma lista com todos os ficheiros no sistema. Os ficheiros desta lista estão identificados com um círculo colorido indicando o seu estado para o utilizador conectado: a amarelo exercício por resolver e a verde exercício já resolvido. Para iniciar o jogo basta selecionar o ficheiro da lista pretendido e escolher a opção "Start".

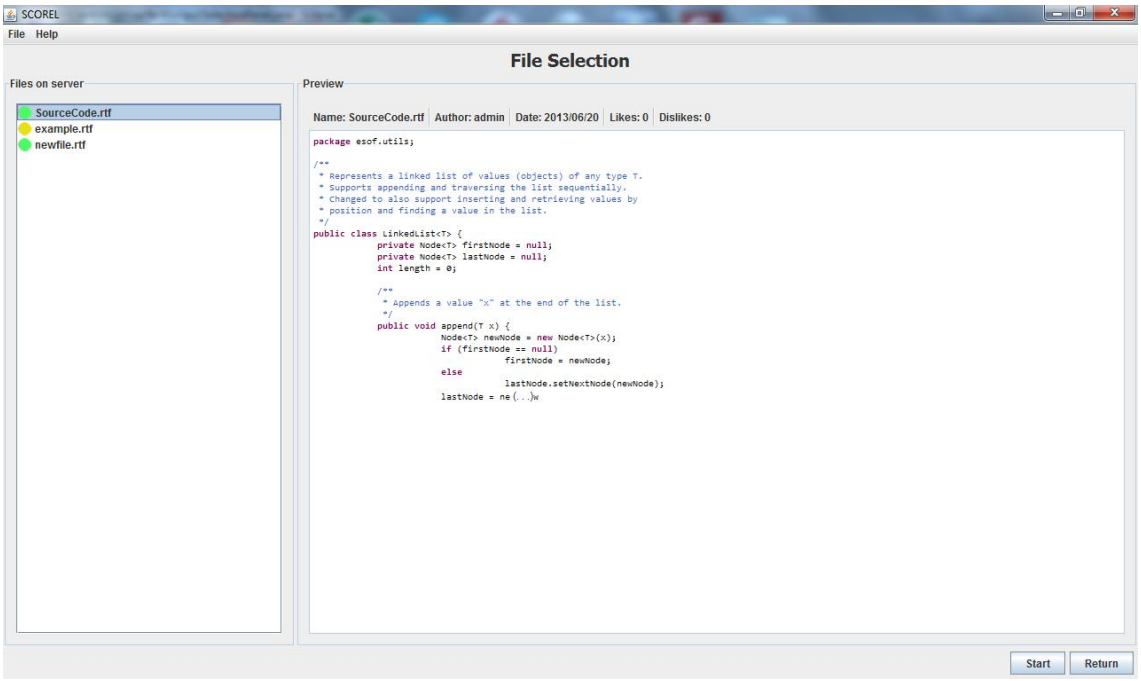

Ilustração 24 Painel de seleção de ficheiro

Uma vez em jogo o utilizador, à semelhança do que acontece quando se carrega um ficheiro, apenas tem que marcar os erros selecionando o texto em questão e com o botão direito escolher o tipo de erro com base na *checklist* associada ao ficheiro. A anulação de erros é feita da mesma forma que no carregamento de ficheiros como se pode ver na Ilustração 25 que representa a área de jogo.

Ilustração 25 Painel de jogo

Após terminar o jogo surge o painel de resultados como apresenta a Ilustração 26. Neste painel é possível ver estatísticas do ficheiro acima e estatísticas do exercício efetuado. Explicando a validade do resultado final obtido, pode-se ver que o utilizador acertou em metade dos erros presentes no exercício, não assinalando nenhum mal. Desta forma daria um resultado de 50% mas teve um total de 58.6% pois resolveu o exercício num tempo muito reduzido. O utilizador pode ainda ver a solução se assim o desejar antes de concluir o jogo, para isso basta escolher a opção “Solution”.

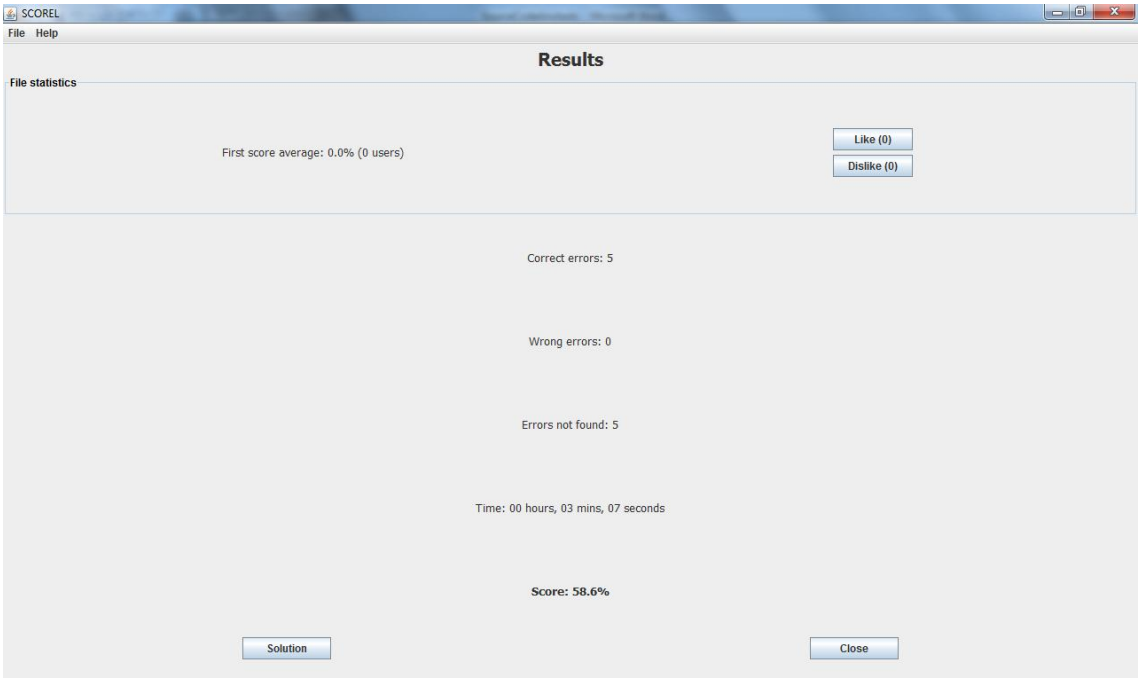

Ilustração 26 Painel de resultados

O painel de soluções apresenta a amarelo, os erros marcados no exercício e a verde as soluções. Através do painel esquerdo, o utilizador pode navegar entre as soluções e ver o tipo e comentário associado a cada um para perceber melhor a solução. Como se pode ver na Ilustração 27, a aplicação assume como correto mesmo que se selecione espaços em branco antes ou depois da seleção correta. Isto acontece no erro em segundo lugar na lista em que a solução é "int length = 0;" e o utilizador selecionou a linha toda. Na secção 6.2 será explicado em detalhe qual o padrão de seleção para cada tipo de erro admitido pela aplicação.

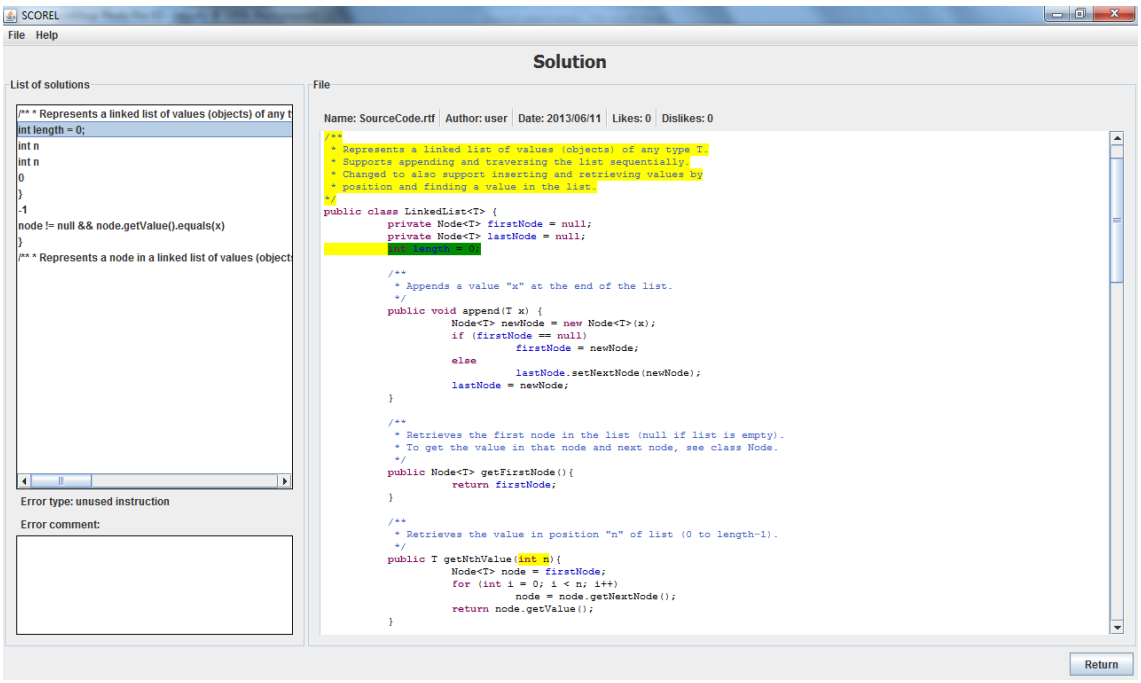

Ilustração 27 Painel de soluções

Após terminar o exercício o utilizador pode confirmar a alteração na lista de melhores resultados como é apresentado na Ilustração 28. Para isso deve selecionar a opção "High Scores" do painel representado pela Ilustração 17. A tabela de melhores resultados apresenta uma lista ordenada pela coluna "Overall Score" que é uma pontuação baseada na média das pontuações obtidas, valor da coluna "Average Score", mas tendo também em conta o número de ficheiros resolvidos.

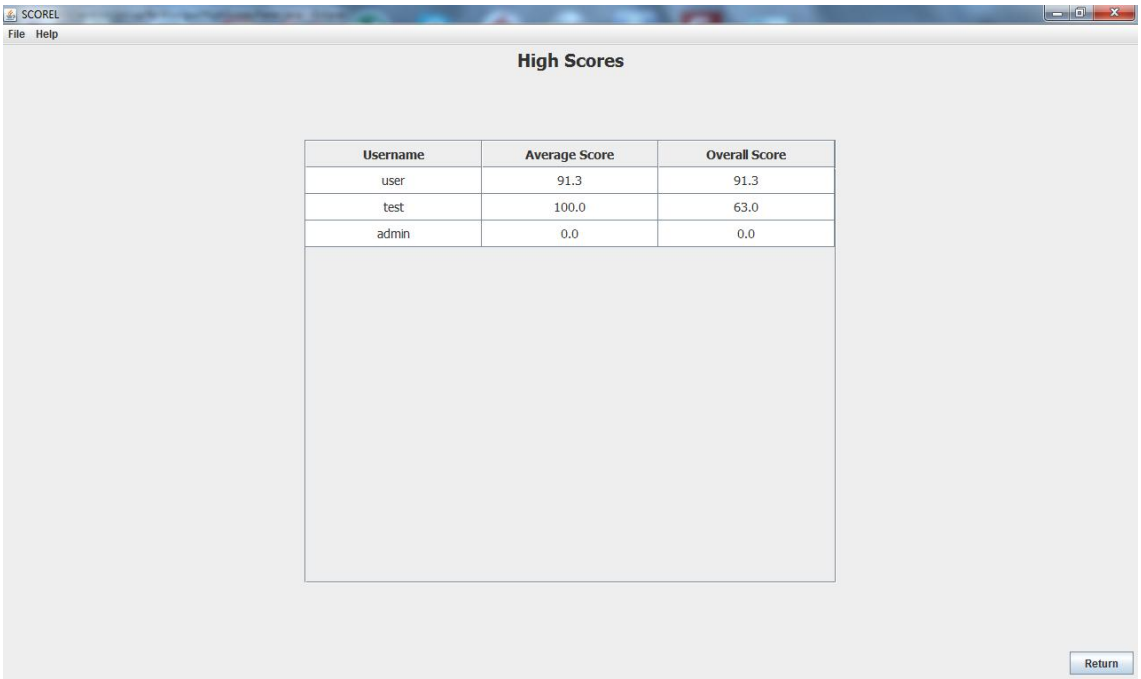

Ilustração 28 Painel de melhores resultados

6.2. Experimentação

A fim de comprovar a utilidade da aplicação foi feito um teste utilizando um exemplo realista usualmente abordado nas aulas de Engenharia de *Software*. O exemplo utilizado encontra-se disponível para consulta em Anexo A - Exemplo utilizado nas aulas de Engenharia de *Software*. Uma vez que a seleção de erros pode, por vezes, ser ambígua, aproveita-se a apresentação deste caso para explicar qual o padrão de seleção de erros que deve ser utilizado para o bom funcionamento da aplicação. É importante referir que de forma a reforçar o sistema de avaliação, a aplicação não considera caracteres em branco nos extremos da *string* para que não se corra o risco de selecionar um espaço em branco por distração e, desta forma, avaliar o erro como mal assinalado. Segue-se então a apresentação da experimentação.

O primeiro erro existente no exemplo é um erro do tipo "**comentário incorreto**" uma vez que falta o autor e a data. A forma correta de o assinalar é selecionando o comentário todo.

```
/**
 * Represents a linked list of values (objects) of any type T.
 * Supports appending and traversing the list sequentially.
 * Changed to also support inserting and retrieving values by
 * position and finding a value in the list.
*/
```

O segundo é um erro do tipo "**instrução não utilizada**". A sua marcação deve abranger toda a instrução em questão.

```
int length = 0;
```

O terceiro erro a aparecer no código é um erro do tipo "**validação de argumento incorreta ou em falta**" pois os limites de "n" não são verificados. Nesta situação o utilizador deve selecionar o tipo e o nome da variável como é apresentado a seguir.

```
public T getNthValue(int n){
```

O quarto erro é do mesmo tipo que o terceiro ou seja "**validação de argumento incorreta ou em falta**" sendo por isso marcado do mesmo modo.

```
public void insert(T x, int n){
```

O quinto erro é um erro do tipo "**dados incorretos**" pois o valor inicial de "i" devia ser 1. Neste caso deve-se selecionar apenas o valor em questão como é mostrado a seguir.

```
for (int i = 0; i < n; i++)
```

O sexto erro do código é um erro do tipo "**pós-condição incorreta ou em falta**". Neste caso falta atualizar o campo "this.lastNode" se ficar no fim da lista e o campo "newNode.nextNode". Este tipo de erros deve ser assinalado na chaveta que fecha a função em causa.

```
public void insert(T x, int n){
```

```
        Node<T> newNode = new Node<T>(x);
        if (n == 0) {
            newNode.setNextNode(firstNode);
            firstNode = newNode;
        }
        else {
            Node<T> prev = firstNode;
            for (int i = 0; i < n; i++)
                prev = prev.getNextNode();
            prev.setNextNode(newNode);
        }
    }
```

O sétimo erro é outro erro do tipo "**dados incorretos**". Como dito acima deve ser marcado apenas no valor em questão.

```
        int index = -1;
```

O oitavo erro é um erro do tipo "**expressão lógica incorreta**" sendo marcada toda a expressão lógica mesmo que só parte dela esteja errada. Neste caso, o correto seria a negação da segunda parte da expressão.

```
        while (node != null && node.getValue().equals(x)) {
```

O nono erro é um erro do tipo "**valor de retorno incorreto**". Este tipo de erros deve ser marcado na chaveta que fecha a função em causa.

```
    public int find(T x){
        int index = -1;
        Node<T> node = firstNode;
        while (node != null && node.getValue().equals(x)) {
            node = node.getNextNode();
            index++;
        }
        return index;
    }
```

O décimo erro é do mesmo tipo que o primeiro, ou seja, "**comentário incorreto**" e deve ser marcado selecionando todo o comentário. Neste caso falta o autor e data.

```
/**
 * Represents a node in a linked list of values (objects) of
type T.
*/
```

Para além dos erros presentes neste exemplo existem ainda outros tipos de erros suportados pela aplicação. Um deles é "**instrução em falta**" em que deve ser marcada a linha completa imediatamente a seguir à posição da instrução em falta e o outro é "**pré-condição incorreta ou em falta**" que à semelhança da pós-condição deve ser assinalada na chaveta de abertura da função.

7. Conclusões e Trabalho Futuro

Este capítulo apresenta conclusões obtidas do trabalho desenvolvido assim como planos de melhoria para futuro.

7.1. Satisfação dos Objetivos

Após o desenvolvimento deste projeto de dissertação pode-se concluir que os objetivos pretendidos foram alcançados. A aplicação obtida no final deste projeto, desenvolvida com a finalidade de permitir praticar técnicas de revisão de código com melhores resultados que o estudo tradicional cumpre os objetivos estipulados inicialmente. E tanto quanto é do conhecimento do autor não há nenhuma outra aplicação do género da desenvolvida. Esta aplicação permite exercitar os tradicionais processos de revisão de código através de um sistema simples e prático, económico, eficiente e comunitário. Exemplo disso é que, utilizando esta aplicação, o utilizador apenas tem que escolher o exercício pretendido e sublinhar os erros marcando-os com um simples clique do rato de forma tão simples e prática como soa. Para além disso, é mais económico pois não tem gastos em material como papel ou instrumento de escrita, é mais eficiente na medida em que poupa tempo na correção do exercício e comunitário pois permite a partilha de exercícios entre utilizadores poupando tempo na pesquisa de novos exercícios.

No fundo, esta aplicação segue a tendência do mundo atual em utilizar as capacidades da tecnologia para complementar ou substituir atividades tradicionais. Um exemplo que se assemelha a este são os exercícios de exames de código da estrada. Toda a gente sabe que existem exercícios em formato digital (cd/dvd) e em formato físico (livro), no entanto a percentagem de pessoas que utiliza o formato digital é superior à de formato físico. Uma das várias vantagens da utilização do formato digital é o simples facto de no computador, o

resultado ser apresentado de imediato e o utilizador não ter de perder tempo a verificar as soluções como o tem de fazer em papel.

É importante realçar que o trabalho obrigou a algumas iterações para afinar a abordagem do ponto de vista funcional e para responder aos principais desafios, crendo-se que a solução final, eventualmente com algumas adaptações, possa vir a ser utilizada nas aulas práticas de Engenharia de *Software*.

7.2. Trabalho Futuro

A aplicação desenvolvida, apesar de satisfazer os objetivos propostos inicialmente, ficou aquém de todas as ideias surgidas. De entre todas as ideias possuídas pretende-se, nesta secção, realçar as que poderiam ter maior influência na melhoria.

Como melhoramento do sistema de *checklist* configurável estava idealizado incluir um sistema de tutorial para a identificação de erros consoante o tipo. Ou seja, o utilizador ao fazer o carregamento de um ficheiro com uma *checklist* configurável teria também de apresentar uma espécie de tutorial de marcação dos tipos de erros dessa *checklist*. Atualmente parte-se do princípio que todos os tipos de erros suportados pela aplicação estejam documentados na secção 6.2.

Uma ideia para desenvolvimento futuro seria a inclusão da atividade de *Team Inspection* que no fundo tratar-se-ia de uma versão da aplicação em modo multijogador na qual os utilizadores poderiam competir com outras equipas enquanto interagiam com a sua através de uma janela de conversação.

Uma outra ideia e bem mais desafiante que a anterior seria o melhoramento do sistema de correção de exercícios passando a ser possível uma avaliação mais precisa através da secção de comentários. Deste modo permitia que o utilizador identificasse não só o tipo de erro assim como a correção para o mesmo tal como pode ser feito no ensino tradicional.

Num cenário mais avultado do projeto passaria por melhorar a aplicação ao ponto de incluir o maior número de atividades possíveis associadas ao estudo de Engenharia de *Software* e integrar a aplicação no moodle da faculdade.

Para trabalho futuro, fica também pendente, a experimentação envolvendo pessoas externas ao trabalho, nomeadamente, alunos de Engenharia de *Software*, que só não foi possível devido a limitações de tempo.

8. Referências

Alice. 2013. «Alice.org». Acedido a 6 de Fevereiro de 2013. http://www.alice.org/.

Arvers, Isabelle. 2009. «Serious Games». *The International Digital Art Magazine*.

Bourque, Pierre, Robert Dupuis, Alain Abran, James W Moore, e Leonard Tripp. 2004. *Guide to the Software Engineering Body of Knowledge*. *IEEE Software*. Los Alamitos, CA, United States.

Docentes de Sistemas Distribuídos, DEI, IST, UTL. 2013. «JAX-WS». Acedido a 20 de Março de 2013. http://disciplinas.ist.utl.pt/~leic-sod.daemon/2012-2013/labs/lab3/jax-ws/index.html.

Frye, Jonathan. 2010. «Understanding "Serious" Games».

Game-Based Learning Dev. 2013. «Game-Based Learning for Software Engineering». *WordPress.com*. Acedido a 5 de Fevereiro de 2013. http://learninggamedev.wordpress.com/2010/01/11/game-based-learning-for-software-engineering/.

MO-SEProcess. 2013. «Software Engineering Process Game - VITAL Lab Public Wiki». Acedido a 6 de Fevereiro de 2013. http://vital.cs.ohiou.edu/vitalwiki/index.php/Software_Engineering_Process_Game.

New Media Institute. 2013. «Game-Based Learning: What It Is, Why It Works, and Where It's Going». Acedido a 5 de Fevereiro de 2013. http://www.newmedia.org/game-based-learning--what-it-is-why-it-works-and-where-its-going.html.

Prensky, Marc. 2001a. «Digital Natives, Digital Immigrants Part 1». *On the Horizon* 9 (5): 1–6.

———. 2001b. «The Digital Game-Based Learning Revolution». Em *Digital Game-Based Learning*, Chapter 1:1–19.

Ranking Web of Universities. 2013. «University | Ranking Web of Universities». Acedido a 15 de Janeiro de 2013. http://www.webometrics.info/en/detalles/up.pt.

Robocode. 2013. «Robocode Home». Acedido a 6 de Fevereiro de 2013. http://robocode.sourceforge.net/.

Serious Game University. 2013. «Serious Game Types & Examples». Acedido a 4 de Fevereiro de 2013. http://www.seriousgameuniversity.com/index.php?option=com_content&view=article&id=22&Itemid=149.

SimSE. 2013. «SimSE Online». Acedido a 6 de Fevereiro de 2013. http://www.ics.uci.edu/~emilyo/SimSE/index.html.

Software Testing Fundamentals. 2013. «Verification Vs Validation». Acedido a 20 de Maio de 2013. http://softwaretestingfundamentals.com/verification-vs-validation/.

Sommerville, Ian. 2011. «Software Engineering » Boston: : Pearson, .

Starting Point-Teaching Entry Level Geoscience. 2013. «Game-Based Learning». Acedido a 4 de Fevereiro de 2013. http://serc.carleton.edu/introgeo/games/index.html.

Wikipédia - Jogo. 2013. «Jogo – Wikipédia, a Enciclopédia Livre». Acedido a 20 de Maio de 2013. http://pt.wikipedia.org/wiki/Jogo.

Wikipédia - Serious game. 2013. «Serious Game – Wikipédia, a Enciclopédia Livre». Acedido a 4 de Fevereiro de 2013. http://pt.wikipedia.org/wiki/Serious_game.

Anexo A - Exemplo utilizado nas aulas de Engenharia de *Software*

LinkedList.java

```
package esof.utils;

/**
 * Represents a linked list of values (objects) of any type T.
 * Supports appending and traversing the list sequentially.
 * Changed to also support inserting and retrieving values by
 * position and finding a value in the list.
*/
public class LinkedList<T> {
    private Node<T> firstNode = null;
    private Node<T> lastNode = null;
    int length = 0;

    /**
     * Appends a value "x" at the end of the list.
     */
    public void append(T x) {
        Node<T> newNode = new Node<T>(x);
        if (firstNode == null)
            firstNode = newNode;
        else
            lastNode.setNextNode(newNode);
        lastNode = newNode;
    }

    /**
     * Retrieves the first node in the list (null if list is
empty).
```

```
     * To get the value in that node and next node, see class
Node.
     */
    public Node<T> getFirstNode(){
        return firstNode;
    }

    /**
     * Retrieves the value in position "n" of list (0 to length-
1).
     */
    public T getNthValue(int n){
        Node<T> node = firstNode;
        for (int i = 0; i < n; i++)
            node = node.getNextNode();
        return node.getValue();
    }

    /**
     * Inserts value "x" in position "n" of the list (0 to
length).
     */
    public void insert(T x, int n){
        Node<T> newNode = new Node<T>(x);
        if (n == 0) {
            newNode.setNextNode(firstNode);
            firstNode = newNode;
        }
        else {
            Node<T> prev = firstNode;
            for (int i = 0; i < n; i++)
                prev = prev.getNextNode();
            prev.setNextNode(newNode);
        }
    }

    /**
     * Retrieves the position of the first occurrence of value
"x"
     * in the list (between 0 and length), or -1 if not found.
     */
    public int find(T x){
        int index = -1;
        Node<T> node = firstNode;
        while (node != null && node.getValue().equals(x)) {
            node = node.getNextNode();
            index++;
        }
        return index;
    }
 }
```

Node.java

```java
package esof.utils;

/**
 * Represents a node in a linked list of values (objects) of
type T.
*/

public class Node<T> {
    private T value;
    private Node<T> nextNode;

    /**
     * Creates a new node containing a value "x".
     * Should only be called from LinkedList.
     */
    Node(T x) {
        value = x;
        nextNode = null;
    }

    /**
     * Sets the next node.
     * Should only be called from LinkedList.
     */
    void setNextNode(Node<T> nextNode) {
        this.nextNode = nextNode;
    }

    /**
     * Retrieves the value stored in this node.
     */
    public T getValue(){
        return value;
    }

    /**
     * Retrieves the next node (null if there is none).
     */
    public Node<T> getNextNode(){
        return nextNode;
    }
}
```